SAS® GLOBAL FORUM 2020 #SASGF

Paper 5172- 2020

RegExing in SAS® for Pattern Matching and Replacement

Pratap Singh Kunwar, The EMMES Company, LLC

ABSTRACT

SAS® has numerous character functions which are very useful for manipulating character fields, but knowing Perl Regular Expressions (RegEx) will help anyone implement complex pattern matching and search-and-replace operations in their programs. Moreover, this skill can be easily portable to other popular languages such as Perl, Python, JavaScript, PHP and more.

This presentation will cover the basics of character classes and metacharacters, using them to build regular expressions in simple examples. These samples range from finding simple literals to finding complex string patterns and replacing them, demonstrating that regular expressions are powerful, convenient and easily implemented.

INTRODUCTION

RegEx has been around for a long time, but most SAS programmers do not use it to its full potential. For years, SAS has been mainly used as a tool for statistical analysis with numeric variables. But when we talk about RegEx we are only talking about character variables. SAS has numerous character (string) functions which are very useful for manipulating character fields, and every SAS programmer is generally familiar with basic character functions such as SUBSTR, SCAN, STRIP, INDEX, UPCASE, LOWCASE, CAT, ANY, NOT, COMPARE, COMPBL, COMPRESS, FIND, TRANSLATE, TRANWRD etc. But even though these common functions are very handy for simple string manipulations, they are not built for complex pattern matching and search-and-replace operations.

RegEx is both flexible and powerful and is widely used in popular programming languages such as Perl, Python, JavaScript, PHP, .NET and many more for pattern matching and translating character strings, which means RegEx skills can be easily imported to other languages.

Learning regular expressions starts with understanding the basics of character classes and metacharacters. Becoming skillful on this topic is not hard. RegEx can be intimidating at first, as it is based on a system of symbols (metacharacters) to describe a text pattern to read text, but this should not be a reason for anyone to put it off.

The table below is used to illustrate the complex syntax. This paper will go through actual examples to help understand some of these metacharacters

Regular Expression PRXMATCH (Matching Text)

prxmatch('/^[BDGKMNSTZ]{1}OO[0-9]{3}-\d{2}\s*$/', id);

Syntax Description:

- ^ asserts position at start of the string
- [BDGKMNSTZ]{1}
  - BDGKMNSTZ matches a single character in the list BDGKMNSTZ (case sensitive)
  - {1} Quantifier - Matches exactly one time
- OO matches the characters OO literally (case sensitive)
- [0-9]{3}
  - 0-9 a single character in the range between 0 and 9
  - {3} Quantifier - Matches exactly 3 times
- - matches the character - literally
- \d{2}
  - matches a digit equal to [0-9]
  - {2} Quantifier - Matches exactly 2 times
- \s*
  - matches any whitespace character
  - * Quantifier - Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)
- $ asserts position at the end of the string

Sample match = BOO003-39

Regular Expression PRXCHANGE (Find and Replace)

prxchange('s/\d//',-1, ‘0001000254698ABCD’)

Syntax Description:

- \s flag for replacement
- \d matches a digit (equal to [0-9])
-
- Replacement group //
  - Delete any matched characters
- -1 Apply to all possible matched characters

Output = ABCD

CHARACTERS AND METACHARACTERS

Regular expressions are built up from metacharacters and their power comes from the use of these metacharacters, which allow the matching of types of text and sequences through systemic searches. There are different sets of characters and metacharacters used in Perl regular expressions as listed below.

SPECIAL CHARACTERS	^	Matches the expression to its right at the start of a string
	$	Matches the expression to its left at the end of a string
	.	Matches any character
	| A|B	alternative matching. A|B – Matches A or B. If A matches first, B will not be tried.
	+	Matches the expression to its left 1 or more times (Greedy Matching)
	*	Matches the expression to its left 0 or more times (Greedy Matching)
	?	Matches the expression to its left 0 or 1 time (Greedy Matching)
	If ? is added to qualifiers (+, *, and ? itself) it will perform matches in a non-greedy manner.	
	{m}	Matches the expression to its left m times
	{m,n}	Matches the expression to its left m to n times but not less (Greedy Matching)
	{m,n}?	Matches the expression to its left m times and ignores n (Non-greedy Matching)
	\	Escapes special characters.
^ and $ are anchors as they assert about position in the string thus, they don't consume characters.		
Greedy vs Lazy matching: a* matches a for 0 or more times as many as possible which is default greedy matching, while doing a*? will match as little as possible that is also called lazy matching.		
Escaping Metacharacters: When a metacharacter(s) itself is in the text, then the metacharacter needs to "escape" from its metacharacter to literal meanings. This is done by putting a backslash in front of it for its literal meaning.		
\. \? * \+ \[\] \\| \(\) \{ \} \$ \^ \\		

CHARACTER CLASSES []	[] contains a set of character to match.	
	[abc]	a, b, or c but does not match abc.
	[^abc]	any but not a, b, or c
	[a-zA-Z]	character between a to z or A to Z. - is used to indicate range of characters
	[a\-z]	matches a, -, or z. It matches – because of having escape char in front of it
	[a-]	Matches a or -
	[-a]	Matches - or a
	[0-9]	any digits
	[(+*)]	Matches (, +, *, or)
Metacharacter inside the []: only the metacharacters that have special meaning for it must be escaped, i.e. ^ after [, - in middle, [and], otherwise no need to use escape character to its left.		
PREDEFINED CHARACTER CLASSES	This offer convenient shorthand for commonly used regular expressions	
	\w	Matches alphanumeric characters a-z, A-Z and 0-9. This also matched underscore _. Equivalent to [a-zA-Z_0-9]
	\d	Matches digits 0-9 Equivalent to [0-9]
	\s	Matches white space character Equivalent to [a-zA-Z_0-9]
	\b	Matches the boundary at the start and end of a word Between \w and \W.
	\W	Matches anything other than a-z, A-Z, 0-9 and _ Equivalent to [^\w]
	\D	Matches any non-digits (anything other than 0-9) Equivalent to [^0-9]
	\S	Matches anything other than whitespace Equivalent to [^\s]
	\B	Matches anything \b does not Equivalent to [a-zA-Z_0-9]
\b and \B also don’t consume characters.		

GROUPS ()	() Matches the expression inside the parentheses and groups it. This is a way to treat multiple characters as a single unit. Groups can be two types Capturing and Non-capturing.	
	(abc)	Capturing Groups take multiple tokens together and create a group for extracting a substring or using a backreference. So it not only matches text but also holds buffer for further processing through extracting, replacing or backreferencing.
	(\w+)\s(\w+)	This represents two groups can be denoted by \$1, \$2 etc. s/(\w+)\s(\w+)/\\$2, \$1\ Having \$2, \$1 will switch words separated by comma i.e. John Smith to Smith, John. Having just \$1 instead of \$2, \$1 will capture will \$1(first name) John. This is an example of extracting and replacing captured group. This is like memory outside match
	(\w)o\1	Matches the results of a capture group (\w). This syntax will match words like pop, dod, xox but don’t match words like aoc. \1 is a numeric reference, which denotes the first group from the left, and these internal numbers usually go from 1 to 9. This is an example of backreferencing a captured group. This is like memory within match, remembering the part of the string matched and the \n inside the match recalls the substring. Another example can be (\w+)\s\1, which will match any repeated words like John John or just just.
	(?:abc)	Non-capturing group; groups multiple tokens together without creating a capture group thus they don’t consumer characters.
	ab(?=cd)	Matches the expression ab only if it is followed by cd (Positive lookahead)
	ab(?!cd)	Matches the expression ab only if it is NOT followed by cd (Negative lookahead)
	(?<=cd)ab	Matches the expression ab only if it is followed by cd (Positive lookbehind)
	(?<!cd)ab	Matches the expression ab only if it is NOT followed by cd (Negative lookbehind)
(sas)+ will match SAS, SASSAS but does not match ss or sossa These anchors match positions instead of characters. But if ^ is used inside the bracket like [^abc], it means negation (anything other than a or b or c).		
Flags	/i	modifier makes matching case-insensitive

SAS PRX Functions

Learning RegExing in SAS, the first thing a programmer needs to know is PRX functions syntax, as they look slightly different than other SAS functions.

Find using PRXMATCH: PRXMATCH function can match the location of the search strings in the source strings. It has two parameters: the first is regular expression ID (search string) and second parameter is character string to be searched (source string). Syntax: PRXMATCH(/regular-expression/, source) Ex. prxmatch('/world/', 'Hello world!'); The above example uses the PRXMATCH function to find the position (=7) of the search-string (world) in the source-string (Hello World) Similarly, using SASHELP.CLASS dataset, if we want to find names that contain 'Jane', we can accomplish by having a syntax like below. PRXMATCH('/Jane/', name); This could also have been accomplished easily with using SAS FINDW function or PROC SQL like clause. The aim of learning RegEx is not just for matching simple literals, like above, but more advanced pattern matching and replacement. This is just a simple matching example using RegEx in SAS.	Find and Replace using PRXCHANGE: PRXCHANGE is not only used to find strings but also to replace them, using specified rules. PRXCHANGE expressions allow the programmer to choose part of the text to replace and the rest to keep. SAS has a simple function named TRANWRD which is very handy for a search-and-replace string, but TRANWRD works only with literal characters or words. Syntax: PRXCHANGE(s/regular-expression/replacement-string/, source) Ex. prxchange('s/world/planet/', 1, 'Hello world!'); The above example uses the PRXCHANGE function to replace 'world' in 'Hello world' with 'planet,' resulting in 'Hello planet' If we just want to replace 'Jane' or 'Janet' with 'X', we can easily write a syntax like below. PRXCHANGE('s/Janet?/X/i, name); The syntax starts with PRXCHANGE, followed by '(/s'. In this syntax, 't' in Janet? is optional. Thus, Jane or Janet will be replaced by 'X'; In this above example, ? acted as a metacharacter, which means the preceding character 0 or 1 time. This works the same way as normal "find and replace" without the use of symbols; more like the SAS TRANWRD function.

APPLICATION 1: SIMPLE MATCH

Technical Description:

If variable *text* contains variations of listed terms in macrovaribale *ptlist1* then flag1='X'.

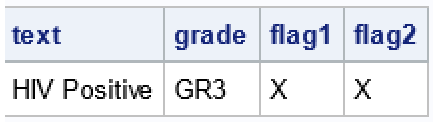
If *grade* contains GR3 or SEVERE then assign variable *flag2*='X'.

Syntax:

```
%let ptlist1=%str(HIVE?|HEPATITIS|HTLV|CYCLOSPORA);
data app1a;
     text="HIV Positive";
     grade='GR3'; output;
run;

data app1;
     set app1a;
     if prxmatch("/(&ptlist1)/", text) then flag1='X';
     if prxmatch("/(GR3|SEVERE)/", grade) then flag2='X';
run;
```

Output:



text	grade	flag1	flag2
HIV Positive	GR3	X	X

Syntax description:

- | Alternation operator
- 1st Alternative HIVE?
  - HIV matches the characters HIV literally (case sensitive)
  - E? matches the character E literally (case sensitive)
  - ? Quantifier — Matches between zero and one times, as many times as possible, giving back as needed (greedy)
- 2nd Alternative HEPATITIS
  - HEPATITIS matches the characters HEPATITIS literally (case sensitive)
- 3rd Alternative HTLV
  - HTLV matches the characters HTLV literally (case sensitive)
- 4th Alternative CYCLOSPORA
  - CYCLOSPORA matches the characters CYCLOSPORA literally (case sensitive)

- 1st Alternative GR3
  - GR3 matches the characters GR3 literally (case sensitive)
- 2nd Alternative SEVERE
  - SEVERE matches the characters SEVERE literally (case sensitive)

APPLICATION 2: IN PROC SQL

Technical Description:

Restrict variable *name* starting with either h or m or j and ending with y (with optional space 0 or more times). Case insensitive.

Syntax:

```
proc sql;
     select *
          from sashelp.class
               where prxmatch('/^(h|m|j).*y\s*$/i', name);
quit;
```

Output:

Name	Sex	Age	Height	Weight
Henry	M	14	63.5	102.5
Jeffrey	M	13	62.5	84
Judy	F	14	64.3	90
Mary	F	15	66.5	112

Syntax description:

- ^ asserts position at start of a line
- 1st Capturing Group (h|m|j)
    - 1st Alternative h
        - h matches the character h literally (case insensitive)
    - 2nd Alternative m
        - m matches the character m literally (case insensitive)
    - 3rd Alternative j
        - j matches the character j literally (case insensitive)
- .* matches any character
    - * Quantifier — Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)
- y matches the character y literally (case insensitive)
    - \s* matches any whitespace character
    - * Quantifier — Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)
- $ asserts position at the end of a line
- i modifier: case insensitive match (ignores case of [a-zA-Z])

APPLICATION 3: BOUNDARY

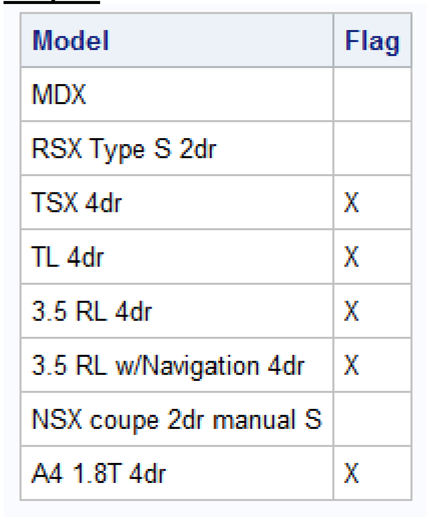
Technical Description:

If variable *Model* contains ‘4dr’ word, then assign variable *Flag*=’X’. Case insensitive.

Syntax:

```
data app3;
    set sashelp.cars (obs=8 keep=model);
    if prxmatch("/\b4dr\b/i", model) then Flag='X';
run;
```

Output:



Model	Flag
MDX	
RSX Type S 2dr	
TSX 4dr	X
TL 4dr	X
3.5 RL 4dr	X
3.5 RL w/Navigation 4dr	X
NSX coupe 2dr manual S	
A4 1.8T 4dr	X

Syntax description:

- \b assert position at a word boundary
- 4dr matches the characters 4dr literally (case insensitive)
- \b assert position at a word boundary
- i modifier: case insensitive match (ignores case of [a-zA-Z])

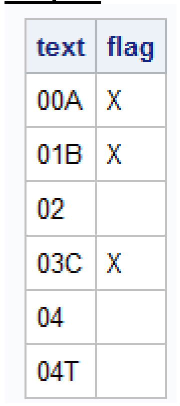
APPLICATION 4: MATCH ENDING CHAR

Technical Description:
If variable *text* start with 0 and ends with either A or B or C then assign *flag*='X'

Syntax:

```
data app4;
      set bb.vismap;
      if prxmatch('/^0\d[A-C]{1}\s*$/i', text) then flag='X';
run;
```

Output:



text	flag
00A	X
01B	X
02	
03C	X
04	
04T	

Syntax description:

- ^ asserts position at start of a line
- 0 matches the characters 0 literally
- \d matches a digit (equal to [0-9])
- [A-C]{1}
  - A-C a single character in the range between A and C (case insensitive)
  - {1} Quantifier — Matches exactly one time
- \s* matches any whitespace character
  - * Quantifier — Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)
- $ asserts position at the end of a line

APPLICATION 5: WITH ALTERNATION (|)

Technical Description:

If macrovariable *text* contains variations of listed terms in macrovariable *idtext* then assign variable *flag* =1. Case insensitive.

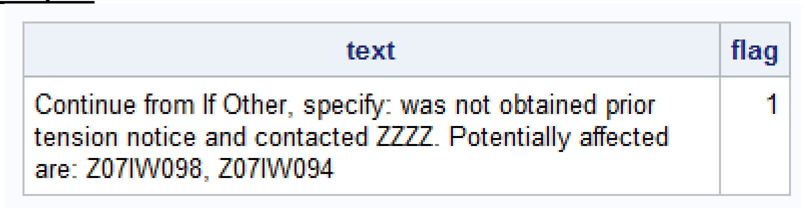
Syntax:

```
%let idtext=%str(Z07IW004|Z07IW005|Z07IW094);

%let text=%str(Continue from If Other, specify: was not obtained prior tension notice and contacted
ZZZZ. Potentially affected are: Z07IW098, Z07IW094);

data app5;
    text="&text";
    if prxmatch("/(&idtext)/", "&text") then flag=1;
run;
```

Output:



text	flag
Continue from If Other, specify: was not obtained prior tension notice and contacted ZZZZ. Potentially affected are: Z07IW098, Z07IW094	1

Syntax description:

- | Alternation operator
- 1st Alternative Z07IW004
    - Z07IW004 matches the characters Z07IW004 literally (case insensitive)
- 2nd Alternative Z07IW005
    - Z07IW005 matches the characters Z07IW005 literally (case insensitive)
- 3rd Alternative Z07IW094
    - Z07IW094 matches the characters Z07IW094 literally (case insensitive)

APPLICATION 6: FIND UPCASE CHAR USING METACHARACTER \U

Technical Description:

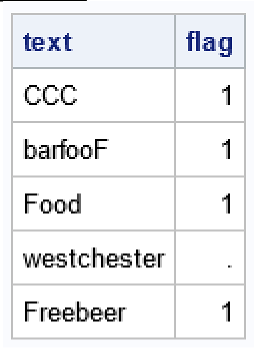
If variable *text* contains any upper-case character than assign variable *flag*=1

Syntax:

```
data app6a;
     input text :$1024.;
     datalines;
CCC
barfooF
Food
westchester
Freebeer
;
run;;

data app6;
     set app6a;
     if prxmatch('/^U[a-z]/', text) then flag=1;
run;
```

Output:



text	flag
CCC	1
barfooF	1
Food	1
westchester	.
Freebeer	1

Syntax description:

- \U specifies that the next string of characters is uppercase.
- [a-z] a single character in the range between a and z
- Equivalent syntax can be /[A-Z]/, without metacharacter \U

APPLICATION 7: VARIOUS ID PATTERN MATCHING

Technical Description:

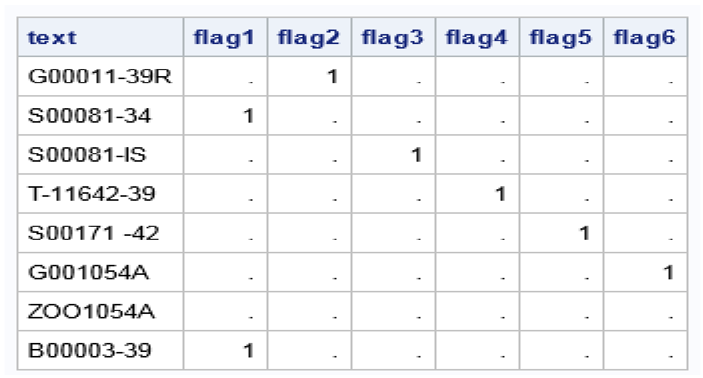
Various types of ID matching.

Syntax:

```
data app7a;
   input text $1-50;
   datalines;
G00011-39R
S00081-34
S00081-IS
T-11642-39
S00171 -42
G001054A
ZOO1054A
B00003-39
;
run;

data app7;
   set app7a;
   if prxmatch( '/^[BDGKMNSTZ]{1}[0-9]{5}-
(\d{1}|C|M)(\d{1}|A)\s*$/', text) then flag1=1;  *S00081-34 or
B00003-39*;
   else if prxmatch('/^[BDGKMNSTZ]{1}[0-9]{5}-(\d{2}[R]{1})\s*$/',
text) then flag2=1;
   else if prxmatch('/^[BDGKMNSTZ]{1}[0-9]{5}-
((\d{2}[\.]{1})|VTM|IS)\s*$/', text) then flag3=1;
   else if prxmatch( '/^[BDGKMNSTZ]{1}( |-)[0-9]{5}-\d{2}\s*$/',
text) then flag4=1;
   else if prxmatch( '/^[BDGKMNSTZ]{1}[0-9]{5} -\d{2}\s*$/', text)
then flag5=1;
   else if prxmatch( '/^[BDGKMNSTZ]{1}[0-
9]{5}(\d{1}|C)(\d{1}|A)\s*$/', text) then flag6=1;
run;
```

Output:



text	flag1	flag2	flag3	flag4	flag5	flag6
G00011-39R	.	1	.	.	.	.
S00081-34	1	.	.	.	.	.
S00081-IS	.	.	1	.	.	.
T-11642-39	.	.	.	1	.	.
S00171 -42	.	.	.	.	1	.
G001054A	.	.	.	.	.	1
ZOO1054A	.	.	.	.	.	.
B00003-39	1	.	.	.	.	.

APPLICATION 8: MULTIPLE SEARCHES

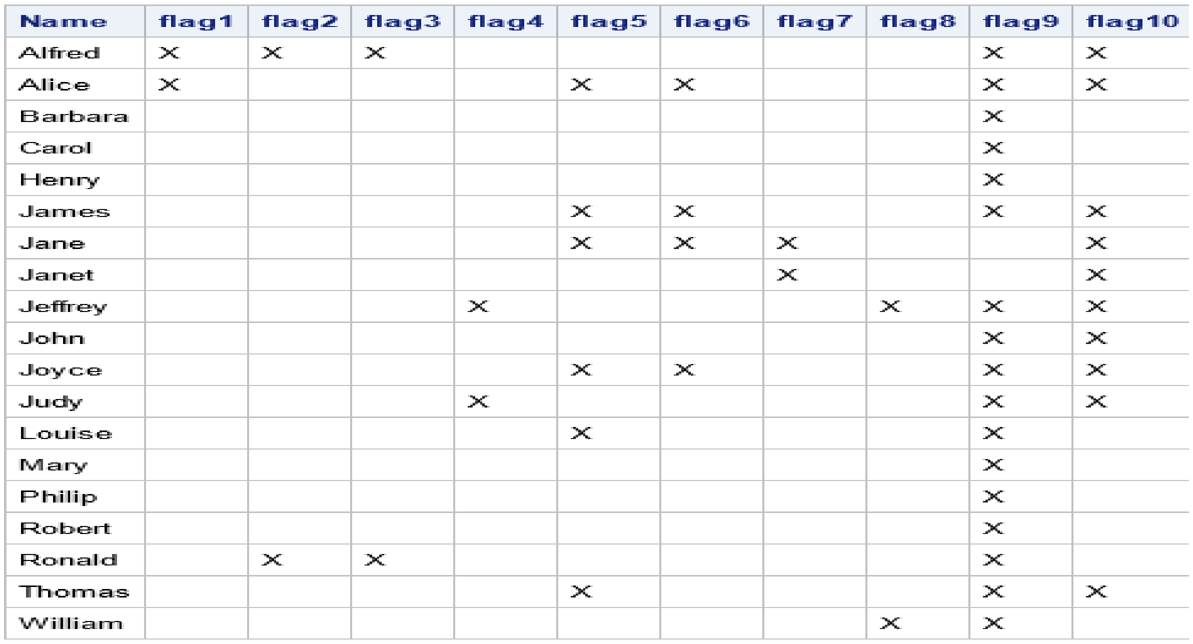
Technical Description:

Various types of matching, see comments after syntax.

Syntax:

```
data app8;
        set sashelp.class;
        if prxmatch ("/^A/", name) then flag1='X'; /*start with A*/
        if prxmatch ("/d$/", strip(name)) then flag2='X'; /*end with d*/
        if prxmatch ("/d\s*$/", name) then flag3='X'; /*end with d or space*/
        if prxmatch ("/^J\w+y\s*$/i", name) then flag4='X'; /*start with J and end with y*/
        if prxmatch("/\w{2}(e|s)\s*$/i", name) then flag5='X'; /*end with e or s*/
        if prxmatch("/^\w{2,4}(e|s)\s*$/i", name) then flag6='X'; /*flag5 but 2 to 4 char)*/
        if prxmatch ("/\Janet?/", name) then flag7='X'; /*ending t is optional*/
        if prxmatch('/(\S)\1/', name ) then flag8='X'; /*2 continious white space*/
        if prxmatch("/[^Janet]/i", strip(name)) then flag9='X';/*Except J|a|n|e|t*/
        if prxmatch("/^[Janet]/i", strip(name)) then flag10='X'; /*start with J|a|n|e|t*/
run;
```

Output:



Name	flag1	flag2	flag3	flag4	flag5	flag6	flag7	flag8	flag9	flag10
Alfred	X	X	X						X	X
Alice	X				X	X			X	X
Barbara									X	
Carol									X	
Henry									X	
James					X	X			X	X
Jane					X	X	X			X
Janet							X			X
Jeffrey				X				X	X	X
John									X	X
Joyce					X	X			X	X
Judy				X					X	X
Louise					X				X	
Mary									X	
Philip									X	
Robert									X	
Ronald		X	X						X	
Thomas					X				X	X
William								X	X	

Syntax description:

- ^A
    - ^ asserts position at start of a line
    - A matches the character A literally (case sensitive)
- d\s*$
    - d matches the character d literally (case insensitive)
    - $ asserts position at the end of a line
- d\s*$
    - d matches the character d literally (case insensitive)
    - \s* matches any whitespace character
    - * Quantifier - Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)
    - $ asserts position at the end of a line
- ^J\w+y\s*$
    - ^ asserts position at start of a line
    - J matches the character J literally (case insensitive)
    - \w+ matches any word character (equal to [a-zA-Z0-9_])
    - + Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)
    - y matches the character y literally (case insensitive)
    - \s* matches any whitespace character
    - * Quantifier — Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)
    - $ asserts position at the end of a line
- \w{2}(e|s)\s*$
    - \w{2} matches any word character (equal to [a-zA-Z0-9_])
    - {2} Quantifier — Matches exactly 2 times
    - 1st Capturing Group (e|s)
        - 1st Alternative e
            - e matches the character e literally (case insensitive)
        - 2nd Alternative s
            - s matches the character s literally (case insensitive)
    - \s* matches any whitespace character
        - * Quantifier — Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)
    - $ asserts position at the end of a line
- ^\w{2,4}(e|s)\s*$
    - ^ asserts position at start of a line
    - \w{2,4} matches any word character (equal to [a-zA-Z0-9_])
    - {2,4} Quantifier — Matches between 2 and 4 times, as many times as possible, giving back as needed (greedy)
    - 1st Capturing Group (e|s)
        - 1st Alternative e
            - e matches the character e literally (case insensitive)
        - 2nd Alternative s
            - s matches the character s literally (case insensitive)
    - \s* matches any whitespace character
        - * Quantifier — Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)
    - $ asserts position at the end of a line
- Janet?
    - Jane matches the characters Jane literally (case insensitive)
    - t? matches the character t literally (case insensitive)

    - ? Quantifier — Matches between zero and one times, as many times as possible, giving back as needed (greedy)
- (\S)\1
    - 1st Capturing Group (\S)
    - \S matches any non-whitespace character
    - \1 matches the same text as most recently matched by the 1st capturing group
- [^Janet]
    - Match a single character not present in the list below [^Janet]
    - Janet matches a single character in the list Janet (case insensitive)
- ^[Janet]
    - ^ asserts position at start of a line
    - Match a single character present in the list below [Janet]
    - Janet matches a single character in the list Janet (case insensitive)

APPLICATION 9: CONDITIONAL REGEX

Technical Description:

If *text* starts with h then match hog else match log or cog (?(?=regex)then|else).

Syntax:

```
data app9a;
   infile datalines truncover;
   input text $ 1-200;
   datalines;
hog
log
cog
zog
;

data app9;
   set app9a;
   newtext= prxmatch('/^(?(?=h)hog|(log|cog))/', text);
run;
```

Output:

text	newtext
hog	1
log	1
cog	1
zog	0

Syntax description:

- ^ asserts position at start of a line
- If Clause (?(?=h)hog|(log|cog))
    - Evaluate the condition below and proceed accordingly
        - Positive Lookahead (?=h)
        - Assert that the Regex below matches
        - h matches the character h literally (case sensitive)
    - If condition is met, match the following regex hog
    - hog matches the characters hog literally (case sensitive)
    - Else match the following regex (log|cog)
        - 1st Capturing Group (log|cog)
            - 1st Alternative log
            - log matches the characters log literally (case sensitive)
        - 2nd Alternative cog
        - cog matches the characters cog literally (case sensitive)

APPLICATION 10: SIMPLE REPLACE

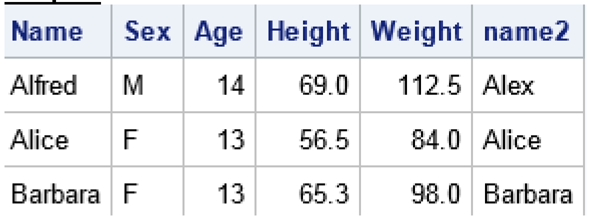
Technical Description:'

If variable *name* equal to ‘Alfred’ then replace it with ‘Alex’.

Syntax:

```
data app10;
    set sashelp.class (obs=3);
    name2=prxchange("s/(Alfred)/Alex/i",-1, name);
run;
```

Output:



Name	Sex	Age	Height	Weight	name2
Alfred	M	14	69.0	112.5	Alex
Alice	F	13	56.5	84.0	Alice
Barbara	F	13	65.3	98.0	Barbara

Syntax description:

- s/ substitution operator
- 1st Capturing Group (Alfred)
  - Alfred matches the characters Alfred literally (case insensitive)
- Replacement group /Alex/
  - Matched Alfred is replaced by Alex (case insensitive)
- i modifier: case insensitive match (ignores case of [a-zA-Z])

APPLICATION 11: INTERCHANGE WORDS

Technical Description:

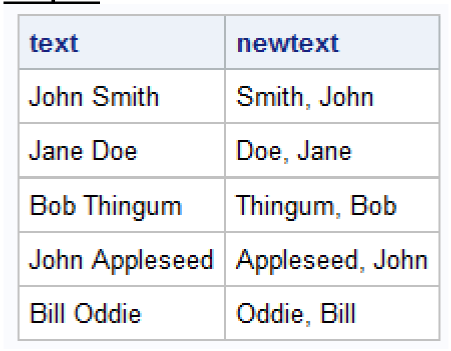
Reverse order of two listed words separated by comma.

Syntax:

```
data app11a;
   input text $1-50;
   datalines;
John Smith
Jane Doe
Bob Thingum
John Appleseed
Bill Oddie
;
```

```
data app11;
      set app11a;
      newtext=prxchange('s/(\w+)\s (\w+)/$2, $1/',  -1,  text);
run;
```

Output:



text	newtext
John Smith	Smith, John
Jane Doe	Doe, Jane
Bob Thingum	Thingum, Bob
John Appleseed	Appleseed, John
Bill Oddie	Oddie, Bill

Syntax description:

- s/ substitution operator
- 1st Capturing Group (\w+)
  - \w+ matches any word character (equal to [a-zA-Z0-9_])
  - + Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)
- \s*? matches any whitespace character
  - *? Quantifier — Matches between zero and unlimited times, as few times as possible, expanding as needed (lazy)
- 2nd Capturing Group (\w+)
  - \w+ matches any word character (equal to [a-zA-Z0-9_])
  - + Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)
- Replacement group \$2, $1\
  - $2 2nd Captured Group
  - $1 1st Captured Group
- -1 Apply to all possible matched characters
- i modifier: case insensitive match (ignores case of [a-zA-Z])

APPLICATION 12: REPLACE WITHIN CHAR CLASS []

Technical Description:

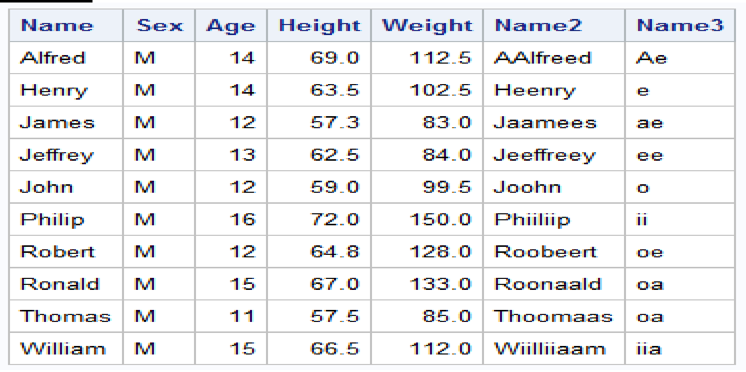
For any vowel characters in variable *name* replace with its double characters in variable *Name2*.

Delete any non-vowel characters from variable *name* in variable *Name3*.

Syntax:

```
data app12;
     set sashelp.class;
     where sex=’M’;
     Name2=prxchange("s/([aeiou])/$1$1/i", -1, name);
     Name3=prxchange("s/([^aeiou])//i", -1, name);
run;
```

Output:



Name	Sex	Age	Height	Weight	Name2	Name3
Alfred	M	14	69.0	112.5	AAlfreed	Ae
Henry	M	14	63.5	102.5	Heenry	e
James	M	12	57.3	83.0	Jaamees	ae
Jeffrey	M	13	62.5	84.0	Jeeffreey	ee
John	M	12	59.0	99.5	Joohn	o
Philip	M	16	72.0	150.0	Phiiliip	ii
Robert	M	12	64.8	128.0	Roobeert	oe
Ronald	M	15	67.0	133.0	Roonaald	oa
Thomas	M	11	57.5	85.0	Thoomaas	oa
William	M	15	66.5	112.0	Wiilliiaam	iia

Syntax description:

- s/ substitution operator
- 1st Capturing Group ([aeiou])
  - Match a single character present in the list below [aeiou]
  - aeiou matches a single character in the list aeiou (case insensitive)

- Replacement group \$1$1\
  - $1 1st Captured Group
  - $1 1st Captured Group

- s/ substitution operator
- 1st Capturing Group ([^aeiou])
  - Match a single character not present in the list below [^aeiou]
  - aeiou matches a single character in the list aeiou (case sensitive)
- Replacement group //
  - Delete any matched characters
- -1 Apply to all possible matched characters

- i modifier: case insensitive match (ignores case of [a-zA-Z])

APPLICATION 13: REMOVE CHAR

Technical Description:

If variable *text* end with either S or T (with optional space 0 or more times), then assign variable *flag*='X'. Remove ending S or T from variable *text* then assign to variable *newtext*.

Syntax:

```
data app13a;
   input text $1-5;
   datalines;
01
01S
02
03S
04
04T
;
data app13;
   set app13a;
   if prxmatch('/[ST]\s*$/i', text) then flag='X';
   newtext=prxchange('s/[ST]\s*$//i', -1, text);
run;
```

Output:

text	flag	newtext
01		01
01S	X	01
02		02
03S	X	03
04		04
04T	X	04

Syntax Description:

- \s flag for replacement
- Match a single character present in the list below [ST]
  - ST matches a single character in the list ST (case insensitive)
- \s* matches any whitespace character
- * Quantifier — Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)
- $ asserts position at the end of a line

- Replacement group //
  - Delete any matched characters
- -1 Apply to all possible matched characters
- i modifier: case insensitive match (ignores case of [a-zA-Z])

APPLICATION 14: REMOVE DIGITS OR LETTERS

Technical Description:

For any digits in variable *alpha*, remove from variable *text*.

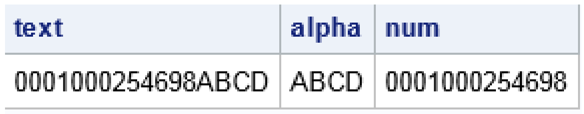
and

Remove any letters from variable *text* in variable *num*.

Syntax:

```
data app14;
   text="0001000254698ABCD";
   alpha=prxchange('s/\d//',-1, text); /*remove digits*/
   num=prxchange('s/[a-z]//i',-1, text); /*remove alphabets*/
run;
```

Output:



```
text               alpha  num
0001000254698ABCD  ABCD   0001000254698
```

Syntax Description:

- \s flag for replacement
- \d matches a digit (equal to [0-9])

- Match a single character present in the list below [a-z]
- a-z a single character in the range between a and z (case insensitive)

- Replacement group //
  - Delete any matched characters
- -1 Apply to all possible matched characters
- i modifier: case insensitive match (ignores case of [a-zA-Z])

APPLICATION 15: REMOVE LEADING ZEROS

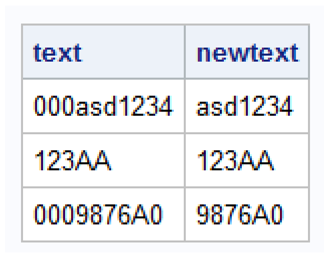
Technical Description:

Remove any leading zeros from variable *text* and assign to new variable *newtext.*

Syntax:

```
data app15a;
    text = '000asd1234'; output;
    text = '123AA'; output;
    text = '0009876A0'; output;
run;
data app15;
    set app15a;
    newtext = prxchange('s/^0+//', -1,  text);
run;
```

Output:



text	newtext
000asd1234	asd1234
123AA	123AA
0009876A0	9876A0

Syntax description:

- s/ substitution operator
- ^ asserts position at start of a line
- 0+ matches the character 0 literally
    - + Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)
- Replacement group //
    - Delete any matched characters
- -1 Apply to all possible matched characters

APPLICATION 16: CHANGING MULTIPLE WORDS TO UPPERCASE USING \U

Technical Description:

Change a few listed words to upper case in new variable *newtext.*

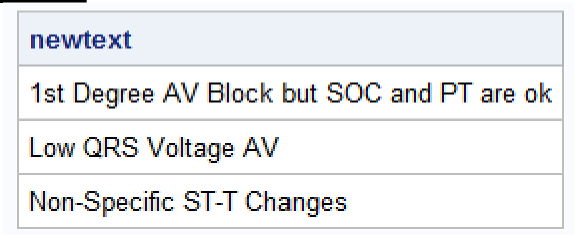
Syntax:

```
data app16a;
     infile datalines truncover;
     input text $ 1-200;
     datalines;
1st Degree Av Block but Soc and Pt are ok
Low Qrs Voltage av
Non-Specific St-T Changes;
run;

data app16;
     set app16a;
     newtext=prxchange('s/(st\-t|qrs|av|soc|pt)/\U$1/i', -1, text);

run;
```

Output:



newtext
1st Degree AV Block but SOC and PT are ok
Low QRS Voltage AV
Non-Specific ST-T Changes

Syntax description:

- s/ substitution operator
- 1st Alternative st\-t
  - o st matches the characters st literally (case insensitive)
  - o \- matches the character - literally (case insensitive)
  - o T matches the character T literally (case insensitive)
- 2nd Alternative qrs
  - o qrs matches the characters qrs literally (case insensitive)
- 3rd Alternative av
  - o av matches the characters av literally (case insensitive)
- 4th Alternative soc
  - o soc matches the characters soc literally (case insensitive)
- 5th Alternative Av
  - o pt matches the characters pt literally (case insensitive)

- Replacement group /\U$1/
  - o \U Transforms next string of characters is uppercase
  - o $1 Captured Group
- -1 Apply to all possible matched characters

APPLICATION 17: CAPTURING PHONE NUMBERS

Technical Description:

Capture various unformatted phone numbers from variable *text* and format them in- (XXX) – XXX – XXXX.

Syntax:

```
data app17a;
     infile datalines truncover;
     input text $ 1-200;
     datalines;
I called Fred at 9:17 am at 785-555-1234 instead of 310.212.3366
10:12 Called George - (913)-555-3213 but he was at 2123639999
816-555-9876 was Irving the time was 1:22 pm
751 555 1212 8384 3:33 Bob
;

data app17;
     set app17a;
          newtext = prxchange('s/\(?(\d{3})\)?[\s-.]?(\d{3})[\s-
.]?(\d{4})/($1) - $2 - $3/', -1, text);

run;
```

Output:

newtext
I called Fred at 9:17 am at (785) - 555 - 1234 instead of (310) - 212 - 3366
10:12 Called George - (913) - 555 - 3213 but he was at (212) - 363 - 9999
(816) - 555 - 9876 was Irving the time was 1:22 pm
(751) - 555 - 1212 8384 3:33 Bob

Syntax description:

- s/ substitution operator
- \(? matches the character (literally
    - ? Quantifier — Matches between zero and one times, as many times as possible, giving back as needed (greedy)
- 1st Capturing Group (\d{3})
    - \d{3} matches a digit (equal to [0-9])
    - {3} Quantifier — Matches exactly 3 times
- \)? matches the character) literally
    - ? Quantifier — Matches between zero and one times, as many times as possible, giving back as needed (greedy)
- Match a single character present in the list below [\s-.]?
    - ? Quantifier — Matches between zero and one times, as many times as possible, giving back as needed (greedy)
    - \s matches any whitespace character

    - -. matches a single character in the list -. (case sensitive)
- 2nd Capturing Group (\d{3})
    - \d{3} matches a digit (equal to [0-9])
    - {3} Quantifier — Matches exactly 3 times
- Match a single character present in the list below [\s-.]?
    - ? Quantifier — Matches between zero and one times, as many times as possible, giving back as needed (greedy)
    - \s matches any whitespace character
    - -. matches a single character in the list -. (case sensitive)
- 3rd Capturing Group (\d{4})
    - \d{4} matches a digit (equal to [0-9])
    - {4} Quantifier — Matches exactly 4 times
- Replacement group / ($1) - $2 - $3/
    - ($1) 1st Captured Group surrounded by ()
    - - $2 - 2nd Captured Group enclosed by -
    - $3 3rd Captured Group
- -1 Apply to all possible matched characters

APPLICATION 18: LOOKAROUND – POSITVE LOOKBEHIND AND POSITVE LOOKAHEAD

Technical Description:

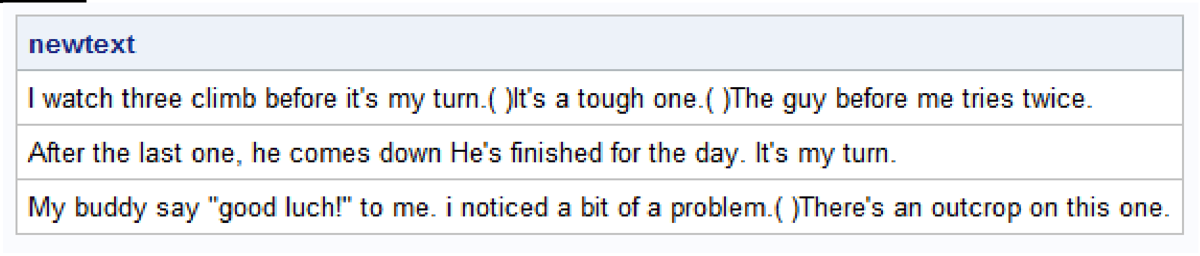
Find any double space after period but before any upper-case char then enclose it with ().

Syntax:

```
data app18a;
      infile datalines truncover;
      input text $ 1-200;
      datalines;
I watch three climb before it's my turn.  It's a tough one.  The guy before me tries twice.
After  the last one, he comes down He's finished for  the day. It's my turn.
My buddy say "good luch!" to me.  i noticed a bit of a problem.  There's an outcrop on this
one.
;

data app18;
      set app18a;
      newtext = prxchange('s/(?<=\.)(\s{2,})(?=[A-Z])/($1)/', -1, text);
run;
```

Output:



newtext
I watch three climb before it's my turn.( )It's a tough one.( )The guy before me tries twice.
After the last one, he comes down He's finished for the day. It's my turn.
My buddy say "good luch!" to me. i noticed a bit of a problem.( )There's an outcrop on this one.

Syntax description:

- s/ substitution operator
- Positive Lookbehind (?<=\.)
  - Assert that the Regex below matches
    - \. matches the character . literally (case sensitive)
- 1st Capturing Group (\s{2,})
  - \s{2,} matches any whitespace character
  - {2,} Quantifier — Matches between 2 and unlimited times, as many times as possible, giving back as needed (greedy)
- Positive Lookahead (?=[A-Z])
  - Assert that the Regex below matches
    - Match a single character present in the list below [A-Z]
    - A-Z a single character in the range between A and Z (case sensitive)
- Replacement group /($1)/
  - ($1) 1st Captured Group surrounded by ()
- -1 Apply to all possible matched characters

APPLICATION 19: REMOVE DUPLICATES USING BACKREFERENCE AND GROUPS

Technical Description:

Remove any repeated words from variable *text*.

Syntax:

```
data app19a;
      infile datalines truncover;
      input text $ 1-200;
      datalines;
It was a CHILLY CHILLY November afternoon, i had JUST JUST
consummated an unusually hearty dinner.
I was sitting alone in the dining-room, with my feet upon the the
fender.
Next WITH WITH some miscellaneous BOTTLES BOTTLES of wine, spirit and
liqueur.;
run;

data app19;
      set app19a;
      newtext =  prxchange('s/\b(\w+)\s\1/$1/', -1,  text);
run;
```

Output:

newtext
It was a CHILLY November afternoon, i had JUST consummated an unusually hearty dinner.
I was sitting alone in the dining-room, with my feet upon the fender.
Next WITH some miscellaneous BOTTLES of wine, spirit and liqueur.

Syntax description:

- s/ substitution operator
- \b assert position at a word boundary
- 1st Capturing Group (\w+)
  - \w+ matches any word character (equal to [a-zA-Z0-9_])
  - + Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)
- \s matches any whitespace character (equal to [\r\n\t\f\v])
- \1 matches the same text as most recently matched by the 1st capturing group
- Replacement group /$1/
  - $1)1st Captured Group
- -1 Apply to all possible matched characters

APPLICATION 20:

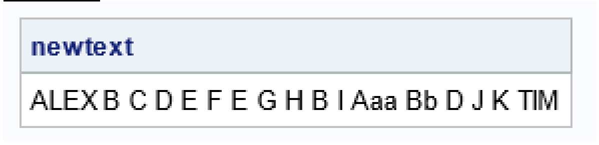
Technical Description:

Remove duplicate words from given list.

Syntax:

```
data app20;
    text = "ALEX ALEX Aaa B C D E F E G H B I Aaa Bb D J K TIM TIM";
        do i = 1 to countw(text);
            newtext = prxchange('s/(\b\w+?\b)(.*?)(?=\b\1{1,}\b)(.?)/$2$3/i',
-1, compbl(text));
        end;
run;
```

Output:



newtext
ALEX B C D E F E G H B I Aaa Bb D J K TIM

Syntax description:

- 1st Capturing Group (\b\w+?\b)
- \b assert position at a word boundary: (^\w|\w$|\W\w|\w\W)
  - \w+? matches any word character (equal to [a-zA-Z0-9_])
  - +? Quantifier — Matches between one and unlimited times, as few times as possible, expanding as needed (lazy)
- \b assert position at a word boundary: (^\w|\w$|\W\w|\w\W)

- 2nd Capturing Group (.*?)
  - .*? matches any character (except for line terminators)
  - *? Quantifier — Matches between zero and unlimited times, as few times as possible, expanding as needed (lazy)
- Positive Lookahead (?=\b\1{1,}\b)
- Assert that the Regex below matches
- \b assert position at a word boundary: (^\w|\w$|\W\w|\w\W)
  - \1{1,} matches the same text as most recently matched by the 1st capturing group
  - {1,} Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)
- \b assert position at a word boundary: (^\w|\w$|\W\w|\w\W)

- 3rd Capturing Group (.?)
  - .? matches any character (except for line terminators)
  - ? Quantifier — Matches between zero and one times, as many times as possible, giving back as needed (greedy)

APPLICATION 21:

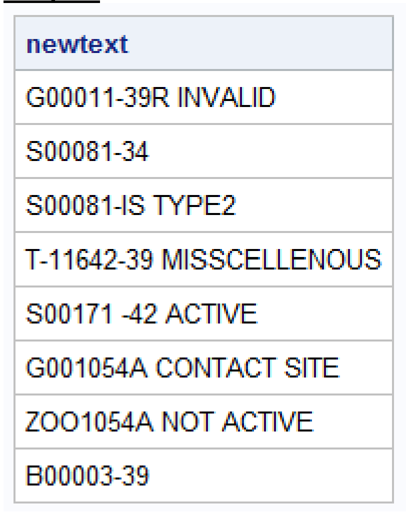
Technical Description:

If text pattern follows syntax ^[BDGKMNSTZ]{1}[0-9]{5}-(\d{2}) then keep only matched pattern otherwise keep entire row.

Syntax:

```
data app21;
   input text $1-50;
   newtext = prxchange('s/^.*?((?(?=.*?(\b(?:^[BDGKMNSTZ]{1}[0-
9]{5}-(\d{2}))\b).*?)\2|.*)).*?$/$1/', -1, text);
   datalines;
G00011-39R INVALID
S00081-34 VALID ID
S00081-IS TYPE2
T-11642-39 MISSCELLENOUS
S00171 -42 ACTIVE
G001054A CONTACT SITE
ZOO1054A NOT ACTIVE
B00003-39 VALID ID
;
run;
```

Output:



newtext
G00011-39R INVALID
S00081-34
S00081-IS TYPE2
T-11642-39 MISSCELLENOUS
S00171 -42 ACTIVE
G001054A CONTACT SITE
ZOO1054A NOT ACTIVE
B00003-39

Syntax description:

- 1st Capturing Group (\b\w+?\b)
- \b assert position at a word boundary: (^\w|\w$|\W\w|\w\W)
    - o \w+? matches any word character (equal to [a-zA-Z0-9_])
    - o +? Quantifier — Matches between one and unlimited times, as few times as possible, expanding as needed (lazy)
- \b assert position at a word boundary: (^\w|\w$|\W\w|\w\W)

- 2nd Capturing Group (.*?)
- ^ asserts position at start of a line
- .*? matches any character (except for line terminators)
  - *? Quantifier — Matches between zero and unlimited times, as few times as possible, expanding as needed (lazy)
- 1st Capturing Group ((?(?=.*?(\b(?:^[BDGKMNSTZ]{1}[0-9]{5}-(\d{2}))\b).*?)\2|.*))
  - If Clause (?(?=.*?(\b(?:^[BDGKMNSTZ]{1}[0-9]{5}-(\d{2}))\b).*?)\2|.*)
    - Evaluate the condition below and proceed accordingly
    - Positive Lookahead (?=.*?(\b(?:^[BDGKMNSTZ]{1}[0-9]{5}-(\d{2}))\b).*?)
    - Assert that the Regex below matches
      - .*? matches any character (except for line terminators)
      - *? Quantifier — Matches between zero and unlimited times, as few times as possible, expanding as needed (lazy)
      - 2nd Capturing Group (\b(?:^[BDGKMNSTZ]{1}[0-9]{5}-(\d{2}))\b)
      - .*? matches any character (except for line terminators)
- If condition is met, match the following regex \2
  - \2 matches the same text as most recently matched by the 2nd capturing group
  - Else match the following regex .*
    - .* matches any character (except for line terminators)
    - * Quantifier — Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)
- .*? matches any character (except for line terminators)
- *? Quantifier — Matches between zero and unlimited times, as few times as possible, expanding as needed (lazy)
- $ asserts position at the end of a line

CONCLUSION

Learning RegEx requires mastering the use of metacharacters, which requires a trial and error approach. Further fine-tuning can be performed by practicing the use of the free online tools listed below or free text editor like Atom in an interactive mode by placing source-string in the text buffer and search-string in the find buffer respectively.

ACKNOWLEDGEMENTS

The authors would like to thank their colleagues at the Emmes Corporation for their feedback and encouragement. The author would also like to thank Ryan Bratt for reviewing this paper.

RECOMMENDED READING

- *https://www.lexjansen.com/*
- https://documentation.sas.com/?docsetId=lefunctionsref&docsetTarget=n0bj9p4401w3n9n1gmv6tfshit9m.htm&docsetVersion=9.4&locale=en#n0jba0adj9x3nyn1v61mcuuhk0xc
- https://documentation.sas.com/?docsetId=lefunctionsref&docsetTarget=n13as9vjfj7aokn1syvfyrpaj7z5.htm&docsetVersion=9.4&locale=en
- https://documentation.sas.com/?docsetId=lefunctionsref&docsetTarget=p1vz3ljudbd756n19502acxazevk.htm&docsetVersion=9.4&locale=en
- https://documentation.sas.com/?docsetId=lefunctionsref&docsetTarget=p0s9ilagexmjl8n1u7e1t1jfnzlk.htm&docsetVersion=9.4&locale=en
- *https://support.sas.com/rnd/base/datastep/perl_regexp/regexp-tip-sheet.pdf*
- *https://www.regular-expressions.info/*
- https://www.rexegg.com/
- https://regex101.com/
- http://regextutorials.com/index.html
- https://regexone.com
- https://regexr.com/

CONTACT INFORMATION

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

Pratap S. Kunwar
The Emmes Company, LLC
401 N Washington St., # 700
E-mail: pkunwar@emmes.com