SESUG Paper 73 - 2025

Exploring the Power of Regular Expressions (RegEx) for Enhanced Pattern Matching and Replacement Operations in SAS®

Pratap Singh Kunwar, The EMMES Company, LLC

ABSTRACT

Regular expressions (RegEx) offer a powerful way to find, extract, and manipulate text patterns—going far beyond what standard SAS® character functions can do. This session is designed for beginner and intermediate SAS users who want to build confidence and skill in applying RegEx to real-world programming challenges.

We'll start with the fundamentals, introducing key concepts like character classes, metacharacters, and quantifiers. From there, we'll explore how to build and use regular expressions for tasks such as validating formats, cleaning messy strings, and performing advanced search-and-replace operations.

Through step-by-step examples, you'll see how RegEx can simplify code, reduce manual text handling, and solve problems that would otherwise require complex logic. You'll also learn tips for troubleshooting patterns and writing expressions that are both powerful and readable.

By the end of the session, you'll have the tools to start using regular expressions confidently in your SAS programs—and transferable knowledge that applies to other languages like Python, Perl, and JavaScript.

INTRODUCTION

Regular expressions (RegEx) have long been a powerful tool for working with text, yet many SAS programmers haven't fully tapped into their potential. While SAS is traditionally known for its strength in statistical analysis and numeric data handling, RegEx shines when working with character variables—especially when tasks go beyond the capabilities of standard SAS functions like SUBSTR, SCAN, INDEX, or CAT.

Unlike these built-in functions, RegEx allows for flexible and precise pattern matching, making it ideal for tasks such as validating formats, cleaning messy strings, and performing advanced search-and-replace operations. Though RegEx syntax may seem intimidating at first—with its use of metacharacters and symbolic patterns—learning the basics is both manageable and rewarding.

RegEx is also widely used in languages like Python, Perl, JavaScript, and .NET, making it a highly transferable skill across programming environments. Once you understand core concepts like character classes, quantifiers, and anchors, you'll be able to write expressions that are both powerful and readable. This session will help you build confidence in using RegEx within SAS, offering practical examples and tips to simplify your code and solve real-world text challenges more efficiently.

REGULAR EXPRESSIONS: CHARACTERS, METACHARACTERS & GROUPS

◆ Metacharacters & Special Symbols

Regular expressions are built using **metacharacters**, which define patterns for matching text. These symbols enable powerful, flexible searches and manipulations.

Symbol & Meaning

Symbol	Meaning
^	Matches the start of a string
$	Matches the end of a string
.	Matches any single character
| A|B	alternative matching. A|B – Matches A or B. If A matches first, B will not be tried.
+	Matches one or more of the preceding element (Greedy)
*	Matches zero or more of the preceding element (greedy)
?	Matches zero or one of the preceding element (greedy)
{m}	Matches exactly *m* times
{m,n}	Matches between *m* and *n* times (greedy)
{m,n}?	Matches between *m* and *n* times (lazy)
\	Escapes a metacharacter to treat it as a literal
Anchors like ^ and $ match positions, not characters—they don't consume characters.	

◆ Greedy vs Lazy Matching

- **Greedy**: Matches as much as possible.
  Example: a* matches all consecutive as.
- **Lazy**: Matches as little as possible.
  Example: a*? matches the shortest possible sequence of as.

◆ Escaping Metacharacters

To match a metacharacter literally (e.g., a period or asterisk), use a backslash:

\. \? * \+ \[\] \| \(\) \{ \} \$ \^ \\

CHARACTER CLASSES []

Character classes match any one character from a set:

Pattern	Matches
[abc]	a, b, or c (not "abc")
[^abc]	Any character except a, b, or c
[a-zA-Z]	Any letter (upper or lower case)
[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 digit
[(+*)]	Matches (, +, *, or)

Note: Inside [], only a few metacharacters need escaping (e.g., ^, -, [,]).

PREDEFINED CLASSES

These are shorthand for common character sets:

Pattern	Matches	Equivalent
\w	Word characters (letters, digits, underscore)	[a-zA-Z0-9_]
\d	Digits	[0-9]
\s	Whitespace	[\t\r\n\f]
\W	Non-word characters	[^\w]
\D	Non-digits	[^\d]
\S	Non-whitespace	[^\s]
\b	Word boundary	
\B	Non-word boundary	

Note: \b and \B are position-based—they don't consume characters.

🔹 Grouping & Backreferences

🔹 Capturing Groups

Parentheses () group expressions and capture matched text for reuse.

Pattern	Description
(abc)	Captures "abc"
(\w+)\s(\w+)	Captures two words separated by space
s/(\w+)\s(\w+)/$2, $1/	Swaps "John Smith" to "Smith, John"

🔹 Backreferences

Use \1, \2, etc., to refer to captured groups:

- (\w)o\1 matches "pop", "dod", "xox"
- (\w+)\s\1 matches repeated words like "John John"

🔹 Non-Capturing Groups

Use (?:...) to group without capturing:

- (?:abc) groups "abc" but doesn't store it.

🔹 Lookahead & Lookbehind

These are zero-width assertions—they match based on context but don't consume characters.

Pattern	Description
ab(?=cd)	Matches "ab" only if followed by "cd" (positive lookahead)
ab(?!cd)	Matches "ab" only if **not** followed by "cd" (negative lookahead)
(?<=cd)ab	Matches "ab" only if preceded by "cd" (positive lookbehind)
(?<!cd)ab	Matches "ab" only if **not** preceded by "cd" (negative lookbehind)

🔹 Flags

Modifiers that change how patterns behave:

- /i — Case-insensitive matching

SAS PRX Functions:

Working with Regular Expressions in SAS: PRX Functions

When applying regular expressions (RegEx) in SAS, the key tools are the **PRX functions**—a specialized set of functions designed to handle pattern matching, extraction, and replacement tasks. These functions have a unique syntax that differs from traditional SAS character functions, and understanding them is essential for leveraging the full power of RegEx in your SAS programs.

- **PRXMATCH: Finding Patterns**

The PRXMATCH function searches for a pattern within a string and returns the position of the first match. It takes two arguments:

1. A regular expression (pattern)
2. The source string to search

Syntax:

PRXMATCH('/pattern/', source)

Example:

prxmatch('/world/', 'Hello world!')

This returns 7, indicating that "world" starts at the 7th character of the string.

Use Case: *To find names containing "Jane" in the SASHELP.CLASS dataset:*

PRXMATCH('/Jane/', name)

While this could also be done using FINDW or PROC SQL LIKE, RegEx becomes especially valuable for more complex pattern matching beyond simple literals.

- **PRXCHANGE: Find and Replace**

The PRXCHANGE function allows you to search for patterns and replace them using RegEx rules. Unlike TRANWRD, which only handles literal replacements, PRXCHANGE supports dynamic and conditional replacements.

Syntax:

PRXCHANGE('s/pattern/replacement/', occurrence, source)

Example:

prxchange('s/world/planet/', 1, 'Hello world!')

This replaces "world" with "planet", resulting in "Hello planet".

PRXCHANGE('s/Janet?/X/i', name)

This replaces both "Jane" and "Janet" with "X". The ? metacharacter makes the "t" optional, and the /i flag makes the match case-insensitive.

This demonstrates how RegEx can replicate and extend the functionality of TRANWRD, especially when dealing with variable patterns.

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	.
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*
  - *st matches the characters st literally (case insensitive)*
  - *\- matches the character - literally (case insensitive)*
  - *T matches the character T literally (case insensitive)*
- *2nd Alternative qrs*
  - *qrs matches the characters qrs literally (case insensitive)*
- *3rd Alternative av*
  - *av matches the characters av literally (case insensitive)*
- *4th Alternative soc*
  - *soc matches the characters soc literally (case insensitive)*
- *5th Alternative Av*
  - *pt matches the characters pt literally (case insensitive)*

- *Replacement group /\U$1/*
  - *\U Transforms next string of characters is uppercase*
  - *$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$|\W\w|\w\W)*
  - *\w+? matches any word character (equal to [a-zA-Z0-9_])*
  - *+? Quantifier — Matches between one and unlimited times, as few times as possible, expanding as needed (lazy)*
- *\b assert position at a word boundary: (^\w|\w$|\W\w|\w\W)*

- *2nd Capturing Group (.*?)*
- *^ 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

Mastering RegEx requires a solid understanding of metacharacters, which often involves a trial-and-error approach to achieve the desired results. The nature of regular expressions—where small changes in the pattern can significantly alter the output—makes hands-on practice essential for developing proficiency.

To further hone your skills, it's beneficial to engage in regular practice using various tools and environments. Free online resources provide an excellent starting point, offering interactive platforms where you can experiment with different patterns and receive immediate feedback. These tools allow you to test and refine your expressions, helping you understand how each metacharacter functions in different contexts.

Additionally, using a text editor in interactive mode can be particularly advantageous. By placing your source string in the text buffer and your search pattern in the find buffer, you can see the effects of your RegEx in real-time. This setup not only facilitates a deeper understanding of how your patterns work but also encourages you to make quick adjustments, fine-tuning your expressions with immediate visual confirmation.

Incorporating these practices into your learning routine will help you develop a more intuitive grasp of RegEx, enabling you to apply it effectively in various programming and data manipulation tasks.

ACKNOWLEDGEMENTS

The authors would like to thank their colleagues at the Emmes Corporation for their feedback and encouragement.

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