



SAS[®] Certified Professional Prep Guide: Advanced Programming Using SAS[®] 9.4

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2019. *SAS® Certified Professional Prep Guide: Advanced Programming Using SAS® 9.4*. Cary, NC: SAS Institute Inc.

SAS® Certified Professional Prep Guide: Advanced Programming Using SAS® 9.4

Copyright © 2019, SAS Institute Inc., Cary, NC, USA

ISBN 978-1-64295-691-7 (Hardcover)

ISBN 978-1-64295-467-8 (Paperback)

ISBN 978-1-64295-468-5 (PDF)

ISBN 978-1-64295-469-2 (Epub)

ISBN 978-1-64295-470-8 (Kindle)

All Rights Reserved. Produced in the United States of America.

For a hard copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication, or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a), and DFAR 227.7202-4, and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

October 2019

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

P1:certprpg

Contents

<i>How to Prepare for the Exam</i>	vii
<i>Using Sample Data</i>	xi
<i>Accessibility Features of the Prep Guide</i>	xiii

PART 1 SQL Processing with SAS 1

Chapter 1 • PROC SQL Fundamentals	3
PROC SQL Basics	4
The PROC SQL SELECT Statement	5
The FROM Clause	9
The WHERE Clause	10
The GROUP BY Clause	19
The HAVING Clause	29
The ORDER BY Clause	32
PROC SQL Options	35
Validating Query Syntax	38
Quiz	39
Chapter 2 • Creating and Managing Tables	43
The CREATE TABLE Statement	43
Using the LIKE Clause	48
Using the AS Keyword	49
The INSERT Statement	51
The DESCRIBE TABLE Statement	59
Using Dictionary Tables	60
Chapter Quiz	63
Chapter 3 • Joining Tables Using PROC SQL	65
Understanding Joins	66
Generating a Cartesian Product	66
Using Inner Joins	68
Using Natural Joins	77
Using Outer Joins	78
Comparing SQL Joins and DATA Step Match-Merges	84
Quiz	89
Chapter 4 • Joining Tables Using Set Operators	95
Understanding Set Operators	96
Using the EXCEPT Set Operator	103
Using the INTERSECT Set Operator	109
Using the UNION Set Operator	114
Using the OUTER UNION Set Operator	120
Quiz	123
Chapter 5 • Using Subqueries	131
Subsetting Data Using Subqueries	131
Creating and Managing Views Using PROC SQL	141

Quiz	149
Chapter 6 • Advanced SQL Techniques	155
Creating Data-Driven Macro Variables with PROC SQL	155
Accessing DBMS Data with SAS/ACCESS	161
The FedSQL Procedure	165
Quiz	171
 PART 2 SAS Macro Language Processing	177
Chapter 7 • Creating and Using Macro Variables	179
Introducing Macro Variables	179
The SAS Macro Facility	181
Using Macro Variables	188
Troubleshooting Macro Variable References	190
Delimiting Macro Variable References	193
Quiz	194
Chapter 8 • Storing and Processing Text	197
Processing Text with Macro Functions	198
Using SAS Macro Functions to Manipulate Character Strings	198
Using SAS Functions with Macro Variables	203
Using SAS Macro Functions to Mask Special Characters	207
Creating Macro Variables during PROC SQL Step Execution	214
Creating Macro Variables during DATA Step Execution	217
Referencing Macro Variables Indirectly	226
Quiz	228
Chapter 9 • Working with Macro Programs	231
Defining and Calling a Macro	232
Passing Information into a Macro Using Parameters	237
Controlling Variable Scope	240
Debugging Macros	245
Conditional Processing	247
Iterative Processing	252
Quiz	254
Chapter 10 • Advanced Macro Techniques	259
Storing Macro Definitions in External Files	259
Understanding Session Compiled Macros	261
Using the Autocall Facility	262
Data-Driven Macro Calls	266
Quiz	268
 PART 3 Advanced SAS Programming Techniques	271
Chapter 11 • Defining and Processing Arrays	273
Defining and Referencing One-Dimensional Arrays	273
Expanding Your Use of One-Dimensional Arrays	283
Defining and Referencing Two-Dimensional Arrays	288
Quiz	293

Chapter 12 • Processing Data Using Hash Objects	297
Declaring Hash Objects	297
Defining Hash Objects	300
Finding Key Values in a Hash Object	302
Writing a Hash Object to a Table	304
Hash Object Processing	306
Using Hash Iterator Objects	311
Quiz	314
Chapter 13 • Using SAS Utility Procedures	317
Creating Picture Formats with the FORMAT Procedure	317
Creating Functions with PROC FCMP	328
Quiz	334
Chapter 14 • Using Advanced Functions	337
Using a Variety of Advanced Functions	337
Performing Pattern Matching with Perl Regular Expressions	344
Quiz	355

PART 4 Workbook 359

Chapter 15 • Practice Programming Scenarios	361
Differences between the Workbook and Certification Exam	362
Scenario 1	362
Scenario 2	363
Scenario 3	363
Scenario 4	364
Scenario 5	365
Scenario 6	366
Scenario 7	366
Scenario 8	367
Scenario 9	368
Scenario 10	368

PART 5 Solutions 371

Chapter 16 • Chapter Quiz Answer Keys	373
Chapter 1: PROC SQL Fundamentals	373
Chapter 2: Creating and Managing Tables	374
Chapter 3: Joining Tables Using PROC SQL	375
Chapter 4: Joining Tables Using Set Operators	376
Chapter 5: Using Subqueries	377
Chapter 6: Advanced SQL Techniques	378
Chapter 7: Creating and Using Macro Variables	379
Chapter 8: Storing and Processing Text	380
Chapter 9: Working with Macro Programs	381
Chapter 10: Advanced Macro Techniques	382
Chapter 11: Defining and Processing Arrays	383
Chapter 12: Processing Data Using Hash Objects	384
Chapter 13: Using SAS Utility Procedures	384
Chapter 14: Using Advanced Functions	385

Chapter 17 • Programming Scenario Solutions	387
Scenario 1	388
Scenario 2	389
Scenario 3	390
Scenario 4	392
Scenario 5	393
Scenario 6	395
Scenario 7	396
Scenario 8	398
Scenario 9	399
Scenario 10	400
 Recommended Reading	403
Index	405

Chapter 14

Using Advanced Functions

Using a Variety of Advanced Functions	337
The LAG Function	337
The COUNT/COUNTC/COUNTW Function	340
The FIND/FINDC/FINDW Function	342
Performing Pattern Matching with Perl Regular Expressions	344
A Brief Overview	344
Using Metacharacters	345
Example: Using Metacharacters	346
The PRXMATCH Function	347
The PRXPARSE Function	349
The PRXCHANGE Function	351
Quiz	355

Using a Variety of Advanced Functions

The LAG Function

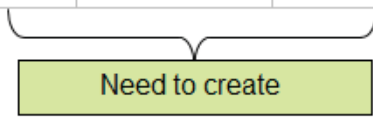
A Brief Overview

Suppose you have the Certadv.Stock6Mon data set that contains opening and closing stock prices for the past six months for two different companies. You are trying to determine which company has the bigger difference in the daily opening price between consecutive days.

To start, consider what you want the LAG function to return, as shown below.

Figure 14.1 Desired LAG Function Results for ABC Company, by Day

Obs	Stock	Date	Open	FirstPrevDay	SecondPrevDay	ThirdPrevDay
1	ABC Company	03/01/2019	54.37	.	.	.
2	ABC Company	03/04/2019	59.53	54.37	.	.
3	ABC Company	03/05/2019	59.45	59.53	54.37	.
4	ABC Company	03/06/2019	57.18	59.45	59.53	54.37
5	ABC Company	03/07/2019	57.55	57.18	59.45	59.53



Need to create

The LAG function enables you to compare the daily opening prices between consecutive days by retrieving the previous values of a column from the last time that the LAG function executed.

LAG Function Syntax

The LAG function retrieves a value from a previous observation. It is able to do so because the function maintains a queue of the previous values. If you use LAG or LAG1, you are looking for the previous value one row back. LAG2 gives you the previous value two rows back. LAG3 gives you the previous value three rows back, and so on. The LAG function is useful for computing differences between rows and computing moving averages.

Syntax, LAG function:

LAG<*n*>(*column*);

n

specifies the number of lagged values.

column

specifies a numeric or character constant, variable, or expression.

Example: Retrieving Previous Values

The following example uses the LAG function to retrieve previous values using assignment statements. The LAG function also creates new variables based on the previous values of Open. Using March 6 as an example, the first previous value is 59.45, the second previous value is 59.53, and the third previous value is 54.37. These values are highlighted in the table below.

Note: For the first observation, there are no previous values to look up, so the assignment statement returns a missing value. For the second observation, there is a first previous value but no second and third previous values, and so on.

```
data work.stockprev;
    set certadv.Stock6Mon(drop=Close);
    FirstPrevDay=lag1(Open);
    SecondPrevDay=lag2(Open);
    ThirdPrevDay=lag3(Open);
run;
proc print data=work.stockprev;
run;
```


Output 14.1 PROC PRINT Output of Work.StockPrev (partial output)

Obs	Stock	Date	Open	FirstPrevDay	SecondPrevDay	ThirdPrevDay
1	ABC Company	03/01/2019	54.37	.	.	.
2	ABC Company	03/04/2019	59.53	54.37	.	.
3	ABC Company	03/05/2019	59.45	59.53	54.37	.
4	ABC Company	03/06/2019	57.18	59.45	59.53	54.37
5	ABC Company	03/07/2019	57.55	57.18	59.45	59.53
6	ABC Company	03/08/2019	60.68	57.55	57.18	59.45
7	ABC Company	03/11/2019	62.50	60.68	57.55	57.18
8	ABC Company	03/12/2019	65.50	62.50	60.68	57.55
9	ABC Company	03/13/2019	65.26	65.50	62.50	60.68
10	ABC Company	03/14/2019	64.56	65.26	65.50	62.50

... more observations ...

Example: Calculating a Moving Average

In addition to computing differences between rows, you can calculate a moving average using the LAG function.

Suppose you have stock prices for the Random Company. The data set contains the opening stock price for the first work day of each month. You need to calculate a moving three-month average. Again, consider what you want the LAG function to return, as shown below.

Figure 14.2 Desired LAG Function Results for Random Company, by Month

Obs	Stock	Date	Open	Open1Month	Open2Month	Open3MonthAvg
1	Random Company	03/01/2019	53.98	.	.	53.98
2	Random Company	04/01/2019	50.39	53.98	.	52.19
3	Random Company	05/01/2019	52.62	50.39	53.98	52.33
4	Random Company	06/03/2019	49.61	52.62	50.39	50.87
5	Random Company	07/01/2019	50.53	49.61	52.62	50.92
6	Random Company	08/01/2019	50.89	50.53	49.61	50.34
7	Random Company	09/03/2019	44.21	50.89	50.53	48.54

Need to create

You can use the LAG function to get the stock price for the past two months. The third row is the first row that calculates an average based on three values.

```
data work.stockavg;
  set certadv.stocks(drop=Close);
  Open1Month=lag1(Open);
  Open2Month=lag2(Open);
  Open3MonthAvg=mean(Open,Open1Month,Open2Month);
```

```

format Open3MonthAvg 8.2;
run;
proc print data=work.stockavg;
run;

```

Output 14.2 PROC PRINT Output of Work.StockAvg

Obs	Stock	Date	Open	Open1Month	Open2Month	Open3MonthAvg
1	Random Company	03/01/2019	53.98	.	.	53.98
2	Random Company	04/01/2019	50.39	53.98	.	52.19
3	Random Company	05/01/2019	52.62	50.39	53.98	52.33
4	Random Company	06/03/2019	49.61	52.62	50.39	50.87
5	Random Company	07/01/2019	50.53	49.61	52.62	50.92
6	Random Company	08/01/2019	50.89	50.53	49.61	50.34
7	Random Company	09/03/2019	44.21	50.89	50.53	48.54

Note: The best practice is to create a lagged value in an assignment statement before using it in a conditional statement.

The COUNT/COUNTC/COUNTW Function

A Brief Overview

Suppose you have the Certadv.Slogans data set, which contains numerous slogans that a company can use for its business. You are asked to identify the number of times a specific word, 24/7, was used in a slogan and how many words were in a slogan. The slogans are separated by commas in a row.

You can use the COUNT function to count the number of times a specific word such as 24/7 appears in the slogan, or you can use the COUNTW function to count the number of words in a slogan. You could even use the COUNTC function to count the number of characters in a slogan.

Note: *Word* is defined as a character constant, variable, or expression.

COUNT/COUNTC/COUNTW Syntax

There are three variations of the COUNT function. Note the slight difference in syntax for the three functions.

Table 14.1 COUNT/COUNTC/COUNTW Syntax

Function Name	Syntax	Function Definition
COUNT	COUNT (string, substring <,modifiers>)	Counts the number of times that a specified substring appears within a character string.
COUNTC	COUNTC (string, character-list <,modifiers>)	Counts the number of characters in a string that appear or do not appear in a list of characters.

Function Name	Syntax	Function Definition
COUNTW	COUNTW (<i>string</i> <,delimiters><,modifiers>)	Counts the number of words in a character string.

character-list

specifies a character constant, variable, or expression that initializes a list of characters. COUNTC counts characters in this list, provided that you do not specify the V modifier in the modifier argument. If you specify the V modifier, all characters that are not in this list are counted. You can add more characters to the list by using other modifiers.

delimiters

can be any of several characters that are used to separate words. You can specify the delimiters by using the *chars* argument, the *modifier* argument, or both.

modifiers

is a character constant, variable, or expression that specifies one or more modifiers. *modifiers* is an optional argument.

i or *I*

ignores the case of the characters. If this modifier is not specified, COUNT counts character substrings only with the same case as the characters in substring.

t or *T*

trims trailing blanks from string, substring, and chars arguments.

string

specifies a character constant, variable, or expression in which substrings are to be counted.

TIP Enclose a literal string of characters in quotation marks.

substring

is a character constant, variable, or expression that specifies the substring of characters to search for in string.

TIP Enclose a literal substring of characters in quotation marks.

Example: Counting the Number of Words

The following example uses the COUNT function to count the number of times 24/7 appears in the Slogans column. The COUNTW function counts the number of words in the Slogans column. The COUNTW function does not specify any delimiter. Therefore, a default list of **blank ! \$ % & () * + , - . / ; < ^ |** is used.

```
data work.sloganact;
  set certadv.slogans;
  Num24=count(Slogans,'24/7');
  NumWord=countw(Slogans);
run;
proc print data=work.sloganact;
run;
```

The COUNT function returns the number of times 24/7 appeared in the Slogan column and assigns the value to Num24. Notice that observation 5 contains 24/365. However, this was not counted as a part of the 24/7. If you change the string to search for to 24/

then 24/365 would appear in the Num24 column. The COUNTW function counts the number of words in each slogan and assigns the value to NumWord.

Output 14.3 PROC PRINT Output of Work.SloganAct

Obs	Slogans	Num24	NumWord
1	repurpose 24/7 markets,productize enterprise web services,brand efficient mindshare	1	11
2	revolutionize killer solutions,expedite e-business e-services,innovate back-end web services	0	13
3	harness 24/7 e-services,redesign visionary systems,exploit strategic schemas	1	11
4	reinvent clicks-and-mortar platforms,revolutionize B2B systems,target integrated models	0	11
5	reintermediate 24/365 systems,cultivate strategic functionalities,brand turn-key synergies	0	11
6	productize best-of-breed communities,benchmark out-of-the-box channels,generate seamless users	0	14
7	iterate killer functionalities,envisioneer user-centric supply-chains,extend end-to-end bandwidth	0	13
8	drive cross-platform portals,embrace clicks-and-mortar infrastructures,target dot-com content	0	13
9	seize holistic web services,harness best-of-breed mindshare,scale integrated synergies	0	12
10	incentivize global niches,generate impactful vortals,aggregate scalable deliverables	0	9

The FIND/FINDC/FINDW Function

A Brief Overview

Suppose you were asked to identify the starting position of the first occurrence of 24/7 in a string. The FIND function finds the starting position of the first occurrence of a substring in a string. Alternatives to the FIND function are the FINDC and FINDW functions, which are also based on finding the first occurrence. The FINDC function returns the starting position where a character from a list of characters is found in a string, and the FINDW function returns the starting position of a word in a string or the number of the word in a string.

FIND/FINDC/FINDW Function Syntax

There are three variations of the FIND function. Note the slight difference in syntax for the three functions.

Table 14.2 FIND/FINDC/FINDW Function Syntax

Function Name	Syntax	Function Definition
FIND	FIND (<i>string</i> , <i>substring</i> <, <i>modifiers</i> >, <i>start-position</i> >);	Searches for a specific substring of characters within a character string. Returns the starting position where a substring is found in a string.
FINDC	FINDC (<i>string</i> , <i>character-list</i> <, <i>modifiers</i> > <, <i>start-position</i> >);	Searches a string for any character in a list of characters. Returns the starting position where a character from a list of characters is found in a string.

Function Name	Syntax	Function Definition
FINDW	FINDW (<i>string</i> , <i>word</i> <, <i>delimiters</i> ><, <i>modifiers</i> > <, <i>start-position</i> >);	Returns the character position of a word in a string, or returns the number of the word in a string.

character-list

is a constant, variable, or character expression that initializes a list of characters.

FINDC searches for the characters in this list, provided that you do not specify the K modifier in the *modifiers* argument. If you specify the K modifier, FINDC searches for all characters that are not in this list of characters. You can add more characters to the list by using other modifiers.

delimiters

can be any of several characters that are used to separate words. You can specify the delimiters by using the *chars* argument, the *modifiers* argument, or both.

modifiers

is a character constant, variable, or expression that specifies one or more modifiers.

i or *I*

ignores the case of the characters. If this modifier is not specified, FIND searches only for character substrings with the same case as the characters in substring.

t or *T*

trims trailing blanks from the *string*, *word*, and *chars* arguments.

start-position

is a numeric constant, variable, or expression with an integer value that specifies the position at which the search should start and the direction of the search.

string

specifies a character constant, variable, or expression that will be searched for substrings.

TIP Enclose a literal string of characters in quotation marks.

substring

is a character constant, variable, or expression that specifies the substring of characters to search for in string.

TIP Enclose a literal substring of characters in quotation marks.

word

is a character constant, variable, or expression that specifies the word to be searched for.

Example: Finding the Word Number

You can use the FINDW function to return the number of the word 24/7 in the Slogans string. The third argument uses a blank to specify the delimiter separating the words in the string. The E modifier tells SAS to count the number of words instead of returning the starting position. The *modifiers* argument must be positioned after the *delimiters* argument. The E modifier is just one of the modifiers that can be used.

```
data work.sloganact;
  set certadv.slogans;
  Num24=count(Slogans,'24/7');
  NumWord=countw(Slogans);
  FindWord24=findw(Slogans,'24/7',' ','e');
```

```
run;
proc print data=work.sloganact;
run;
```

Output 14.4 PROC PRINT Output of Work.SloganAct (partial output)

Obs	Slogans	Num24	NumWord	FindWord24
1	repurpose 24/7 markets,productize enterprise web services,brand efficient mindshare	1	11	2
2	revolutionize killer solutions,expedite e-business e-services,innovate back-end web services	0	13	0
3	harness 24/7 e-services,redesign visionary systems,exploit strategic schemas	1	11	2
4	reinvent clicks-and-mortar platforms,revolutionize B2B systems,target integrated models	0	11	0
5	reintermediate 24/365 systems,cultivate strategic functionalities,brand turn-key synergies	0	11	0
6	productize best-of-breed communities,benchmark out-of-the-box channels,generate seamless users	0	14	0
7	iterate killer functionalities,envisioneer user-centric supply-chains,extend end-to-end bandwidth	0	13	0
8	drive cross-platform portals,embrace clicks-and-mortar infrastructures,target dot-com content	0	13	0
9	seize holistic web services,harness best-of-breed mindshare,scale integrated synergies	0	12	0
10	incentivize global niches,generate impactful vortals,aggregate scalable deliverables	0	9	0

Performing Pattern Matching with Perl Regular Expressions

A Brief Overview

Perl regular expressions enable you to perform pattern matching by using functions. A *regular expression* is a sequence of strings that defines a search pattern.

For example, suppose you have the Certadv.NANumbr data set, which contains phone numbers for the United States, Canada, and Mexico.

Figure 14.3 Certadv.NaNumbr Data Set (partial output)

Obs	Name	PhoneNumber	Country
1	Alexander Mcknight	(738) 766-2114	Canada
2	Alison Campbell	943.519.8369	United States
3	Amador Alvaro Luna	3581599311	Mexico
4	Amanda Johnson	362-686-6286	Canada
5	Amy Williams	953-246-7733	United States
6	Ann Keith	(375) 862-7384	Canada
7	Anne Weaver	793-199-3925	United States
8	Arturo Longoria	203-752-8263	Mexico
9	Brandon Kerr	555-677-4102	United States
10	Camilo Indira Mojica Romero	718.690.4147	Mexico

By using a regular expression, you can find valid values for Phone. The advantage of using regular expressions is that you can often accomplish in only one Perl regular expression function something that would require a combination of traditional SAS functions to accomplish.

In SAS, you use Perl regular expressions within the functions and call routines that start with PRX. The PRX functions use a modified version of the Perl language (Perl 5.6.1) to perform regular expression compilation and matching.

Using Metacharacters

The Perl regular expressions within the PRX functions and call routines are based on using metacharacters. A metacharacter is a character that has a special meaning during pattern processing. You can use metacharacters in regular expressions to define the search criteria and any text manipulations. The following table lists the metacharacters that you can use to match patterns in Perl regular expressions.

Table 14.3 Basic Perl Metacharacters and Their Descriptions

Metacharacter	Description	Example
/.../	Provides the starting and ending delimiter.	s/ ([a-z]) / X / substitutes X in place of a space followed by a lowercase letter and then a space.
(...)	Enables grouping.	f(u boo)bar matches " fubar " or " foobar ".
	Denotes the OR situation.	
\d	Matches a digit (0–9).	\d\d\d\d matches any four-digit string (0-9) such as " 1234 " or " 6387 ".
\D	Matches a non-digit such as a letter or special character.	\D\D\D\D matches any four non-digit string such as " WxYz " or " AVG% ".
\s	Matches a whitespace character such as a space, tab, or newline.	x\sx matches " x x " (space between the letters x) or " x x " (tab between the letters x).
\w	Matches a group of one or more characters (a-z, A-Z, 0-9, or an underscore).	\w\w\w matches any three-word characters.
.	Matches any character.	mi.e matches " mike " and " mice ".
[...]	Matches a character in brackets.	[dmn]ice matches " dice " or " mice " or " nice " \d[6789]\d matches " 162 " or " 574 " or " 685 " or " 999 ".
[^...]	Matches a character not in brackets.	[^] matches [^] matches " " but not " "

Metacharacter	Description	Example
<code>^</code>	Matches the beginning of the string.	<code>d[^a]me</code> matches "dime" or "dome" but not "dame".
<code>\$</code>	Matches the end of the string.	<code>ter\$</code> matches "winter" not "winner" or "terminal".
<code>\b</code>	Matches a word boundary (the last position before a space).	<code>bar\b</code> matches "bar food" but not "barfood" or "barter".
<code>\B</code>	Matches a non-word boundary.	<code>bar\B</code> matches "foobar" but not "bar food".
<code>*</code>	Matches the preceding character 0 or more times.	<ul style="list-style-type: none"> <code>zo*</code> matches "z" and "zoo" <code>*</code> is equivalent to <code>{0,}</code>
<code>+</code>	Matches the preceding character 1 or more times.	<ul style="list-style-type: none"> <code>zo+</code> matches "zo" and "zoo". <code>zo+</code> does not match "z" <code>+</code> is equivalent to <code>{1,}</code>
<code>?</code>	Matches the preceding character 0 or 1 times.	<ul style="list-style-type: none"> <code>do(es)?</code> matches the "do" in "do" or "does" <code>?</code> is equivalent to <code>{0,1}</code>
<code>{n}</code>	Matches exactly <i>n</i> times.	<code>fo{2}bar</code> matches "foobar" but not "fobar" or "fooobar".
<code>\</code>	Overrides the next metacharacter such as a (or ?)	<code>final\.</code> matches "final." "final" is followed by the character '.'

Example: Using Metacharacters

A valid United States, Canada, or Mexico phone number contains a three-digit area code, followed by a hyphen (-), a three-digit prefix, and then the remaining numbers. More specifically, the first digit of the area code and prefix cannot start with 0 or 1.

A Perl regular expression must start and end with a delimiter. The following example uses parentheses to represent a group of numbers that is required. The first two groups specify that first there must be a digit 2 through 9 followed by two more digits. In the last group, there must be four digits. The hyphens between the groups signify the hyphens between the numbers in the output.

```
/([2-9]\d\d)-([2-9]\d\d)-(\d{4})/
```


Output 14.5 *Certadv.NaNumbr Data Set (partial output)*

Obs	Name	PhoneNumber	Country
1	Alexander Mcknight	(738) 766-2114	Canada
2	Alison Campbell	943.519.8369	United States
3	Amador Alvaro Luna	3581599311	Mexico
4	Amanda Johnson	362-686-6286	Canada
5	Amy Williams	953-246-7733	United States
6	Ann Keith	(375) 862-7384	Canada
7	Anne Weaver	793-199-3925	United States
8	Arturo Longoria	203-752-8263	Mexico
9	Brandon Kerr	555-677-4102	United States
10	Camilo Indira Mojica Romero	718.690.4147	Mexico

The PRXMATCH Function

A Brief Overview

The Perl regular expression using metacharacters can be used with the PRX functions. The PRXMATCH function searches for a pattern match and returns the position at which the pattern is found. A value of zero is returned if no match is found. This function has two arguments. The first argument specifies the Perl regular expression that contains your pattern. The second argument is the character constant, column, or expression that you want to search.

PRXMATCH Syntax

Syntax, PRXMATCH function:

PRXMATCH (*Perl-regular-expression*, *source*);

Perl-regular-expression

specifies a character value that is a Perl regular expression. The expression can be referenced using a constant, a column, or a pattern identifier number.

source

specifies a character constant, variable, or expression that you want to search.

Example: PRXMATCH Function Using a Constant

The PRXMATCH function is commonly used for validating data. The following example uses the PRXMATCH function to validate whether a phone number pattern is present.

If the pattern is present, a numeric value is returned to the pattern's starting position. For this example, the pattern was found in 19 rows.

The example specifies the expression as a hard-coded constant as the first argument of the function. When a constant value is specified, the constant must be in quotation marks (either single or double). When you specify the expression as a constant, the expression is compiled once, and each use of the PRX function reuses the compiled expression.

Compiling the expression only once saves time. The compiled version is saved in memory.

```
data work.matchphn;
  set certadv.nanumbr;
  loc=prxmatch('/([2-9]\d\d)-([2-9]\d\d)-(\d{4})/', PhoneNumber);
run;
proc print data=work.matchphn;
  where loc>0;
run;
```

Output 14.6 PROC PRINT Result of Work.MatchPhn

Obs	Name	PhoneNumber	Country	loc
4	Amanda Johnson	362-686-6286	Canada	1
5	Amy Williams	953-246-7733	United States	1
8	Arturo Longoria	203-752-8263	Mexico	1
9	Brandon Kerr	555-677-4102	United States	1
16	Denise Todd	944-905-6288	United States	1
24	Francisco Javier Vanesa Espinoza Pajez	692-804-6430x771	Mexico	1
28	Jaime White	466-646-6557	United States	1
29	Jeffrey Archer	445-765-3784	United States	1
48	Leonor Cisneros	623-656-4441	Mexico	1
50	Lisa Evans PhD	244-697-6738	Canada	1
57	Marissa Hudson	814-917-4811	Canada	1
62	Melissa Gross	879-348-5158	United States	1
65	Minerva Baeza	542-214-2366	Mexico	1
66	Nancy Thomas	642-802-8384	United States	1
69	Pablo Montalvo	630-742-7059x89285	Mexico	1
71	Pedro Vallejo Salgado	875-613-3160	Mexico	1
77	Sarah Young	530-587-5777	United States	1
84	Timothy Christian	862-737-4712	Canada	1
87	William Small	480-398-3374	United States	1

Example: PRXMATCH Function Using a Column

Instead of using the first argument to specify a constant for the regular expression, you can refer to a column that contains the expression. This is a commonly used technique when you might need to manipulate the assignment statement that is specifying the expression.

When the first argument refers to a column instead of a constant, the expression is compiled for each execution of the function. To avoid compiling the expression each time, specify the option of a lower or uppercase O at the end of the expression. This makes SAS compile the expression only once. This is a useful approach when you have large data sets, as it decreases your processing time.

```
data work.phnumbr (drop=Exp);
```

```

set certadv.nanumbr;
Exp=' / ([2-9]\d\d) - ([2-9]\d\d) - (\d{4}) /o';
Loc=prxmatch(Exp,PhoneNumber);
run;
proc print data=work.phnumbr;
  where loc>0;
run;

```

Output 14.7 PROC PRINT Result of Work.PhNumbr

Obs	Name	PhoneNumber	Country	loc
4	Amanda Johnson	362-686-6286	Canada	1
5	Amy Williams	953-246-7733	United States	1
8	Arturo Longoria	203-752-8263	Mexico	1
9	Brandon Kerr	555-677-4102	United States	1
16	Denise Todd	944-905-6288	United States	1
24	Francisco Javier Vanesa Espinoza Pajez	692-804-6430x771	Mexico	1
28	Jaime White	466-646-6557	United States	1
29	Jeffrey Archer	445-765-3784	United States	1
48	Leonor Cisneros	623-656-4441	Mexico	1
50	Lisa Evans PhD	244-697-6738	Canada	1
57	Marissa Hudson	814-917-4811	Canada	1
62	Melissa Gross	879-348-5158	United States	1
65	Minerva Baeza	542-214-2366	Mexico	1
66	Nancy Thomas	642-802-8384	United States	1
69	Pablo Montalvo	630-742-7059x89285	Mexico	1
71	Pedro Vallejo Salgado	875-613-3160	Mexico	1
77	Sarah Young	530-587-5777	United States	1
84	Timothy Christian	862-737-4712	Canada	1
87	William Small	480-398-3374	United States	1

The PRXPARSE Function

A Brief Overview

Another method for specifying the Perl regular expression is to specify a pattern identifier number. Before using PRXMATCH, you can use the PRXPARSE function to create the pattern identifier number. This function references the regular expression either as a constant or a column. The function returns a pattern identifier number. This number can then be passed to PRX functions and call routines to reference the regular expression. It is not required to use the pattern identifier number with the PRXMATCH function, but some of the other PRX functions and call routines do require the pattern identifier number.

PRXPARSE Function Syntax

The PRXPARSE function returns a pattern identifier number that is used by other PRX functions and call routines.

Syntax, PRXPARSE function:

pattern-ID-number=**PRXPARSE** (*Perl-regular-expression*);

pattern-ID-number

is a numeric pattern identifier that is returned by the PRXPARSE function.

Perl-regular-expression

specifies a character value that is a Perl regular expression. The expression can be referenced using a constant, a column, or a pattern identifier number.

Example: PRXPARSE and PRXMATCH Function Using a Pattern ID Number

In this example, the regular expression is being assigned to the column Exp. The PRXPARSE function is referencing this column. Because the expression ends with the O option, the function compiles the value only once. The PRXPARSE function returns a number that is associated with this expression. In this example, the number is a value of 1, and the value is being stored in the Pid column.

PRXMATCH then references this number in the Pid column as its first argument. If the O option had not used at the end of the Perl regular expression, the value of Pid would differ for each row.

```
data work.phnumbr (drop=Exp);
  set certadv.nanumbr;
  Exp='([2-9]\d\d)-([2-9]\d\d)-(\d{4})/o';
  Pid=prxparse(Exp);
  Loc=prxmatch(Pid,PhoneNumber);
run;
proc print data=work.phnumbr;
run;
```

Output 14.8 PROC PRINT Output of Work.PhNumbr (partial output)

Obs	Name	PhoneNumber	Country	Pid	Loc
1	Alexander Mcknight	(738) 766-2114	Canada	1	0
2	Alison Campbell	943.519.8369	United States	1	0
3	Amador Alvaro Luna	3581599311	Mexico	1	0
4	Amanda Johnson	362-686-6286	Canada	1	1
5	Amy Williams	953-246-7733	United States	1	1
6	Ann Keith	(375) 862-7384	Canada	1	0
7	Anne Weaver	793-199-3925	United States	1	0
8	Arturo Longoria	203-752-8263	Mexico	1	1
9	Brandon Kerr	555-677-4102	United States	1	1
10	Camilo Indira Mojica Romero	718.690.4147	Mexico	1	0

The PRXCHANGE Function

A Brief Overview

The PRXCHANGE function performs a substitution for a pattern match. This function has three arguments. The first argument is the Perl regular expression, which can be specified as a constant, a column, or a pattern identifier number that comes from the PRXPARSE function. The second argument is a numeric value that specifies the number of times to search for a match and replace it with a matching pattern. If the value is -1, then the matching pattern continues to be replaced until the end of the source is reached. The third argument is the character constant, column, or expression that you want to search for.

PRXCHANGE Function Syntax

The PRXCHANGE function performs a substitution for a pattern match.

Syntax, PRXCHANGE function:

PRXCHANGE (*Perl-regular-expression, times, source*)

Perl-regular-expression

specifies a character value that is a Perl regular expression. The expression can be referenced using a constant, a column, or a pattern identifier number.

times

is a numeric constant, variable, or expression that specifies the number of times to search for a match and replace a matching pattern.

source

specifies a character constant, variable, or expression that you want to search.

Example: Using the PRXCHANGE Function to Standardize Data

The PRXCHANGE function is commonly used to standardize data. For example, the Certadv.SocialAcct data set contains social media preference data for users between the ages of 18 and 50. The goal is to standardize the Certadv.SocialAcct data set by substituting Facebook for Fb and FB as well as Instagram for IG.

Figure 14.4 *Certadv.SocialAcct (partial output)*

Obs	Name	Age	Social_Media_Pref1	Social_Media_Pref2
1	Emily Stafford	23	IG	FB
2	Rachel Valenzuela	24	IG	FB
3	Roger Kelly	26	IG	FB
4	Laura Ramirez	27	IG	FB
5	Michael Williams	30	IG	FB
6	Danielle Middleton	31	IG	FB
7	Matthew Mcguire	32	IG	FB
8	Natalie Velasquez	33	IG	FB
9	Gary Andrews DVM	34	IG	FB
10	Jeremy Blake	37	IG	FB

When you are writing the Perl regular expression for substitution, start the expression with a lowercase *s*. The lowercase *s* signifies that substitution needs to happen instead of matching.

Following the lowercase *s*, place the beginning delimiter before the forward slash. Also, place the forward slash at the end of the expression. There is another forward slash between the starting and ending forward slashes.

Before the middle forward slash, specify the pattern that you are searching for, enclosed in parentheses. After the middle forward slash, specify the pattern that is to be used for substitution.

In this example, you are looking for the capital letters FB and IG in both `Social_Media_Pref1` and `Social_Media_Pref2` variables. If the pattern is found, then replace with Facebook and Instagram, respectively. The *i* modifier ignores the case of the pattern that you are searching for.

```
data work.prxsocial;
  set certadv.socialacct;
  Social_Media_Pref1=prxchange('s/(FB)/Facebook/i',-1,Social_Media_Pref1);
  Social_Media_Pref1=prxchange('s/(IG)/Instagram/i',-1,Social_Media_Pref1);
  Social_Media_Pref2=prxchange('s/(FB)/Facebook/i',-1,Social_Media_Pref2);
  Social_Media_Pref2=prxchange('s/(IG)/Instagram/i',-1,Social_Media_Pref2);
run;
proc print data=work.prxsocial;
run;
```

Output 14.9 PROC PRINT Output of Work.PrxSocial (partial output)

Obs	Name	Age	Social_Media_Pref1	Social_Media_Pref2
1	Emily Stafford	23	Instagram	Facebook
2	Rachel Valenzuela	24	Instagram	Facebook
3	Roger Kelly	26	Instagram	Facebook
4	Laura Ramirez	27	Instagram	Facebook
5	Michael Williams	30	Instagram	Facebook
6	Danielle Middleton	31	Instagram	Facebook
7	Matthew Mcguire	32	Instagram	Facebook
8	Natalie Velasquez	33	Instagram	Facebook
9	Gary Andrews DVM	34	Instagram	Facebook
10	Jeremy Blake	37	Instagram	Facebook

Example: Changing the Order Using the PRXCHANGE Function

Suppose you have the Certadv.SurvNames data set with names from the self-reported survey. Every 50th surveyor is given a gift card that is to be mailed to the surveyor's home. You are asked to quickly reverse the names of the survey takers. You can use the PRXCHANGE function to reverse the order of the names.

```
data work.revname;
  set certadv.survnames;
  ReverseName=prxchange('s/(\w+), (\w+)/$2 $1/', -1, name);
run;
proc print data=work.revname;
run;
```

Output 14.10 PROC PRINT Result of Work.RevName

Obs	Name	ReverseName
1	Rivera, Marilyn	Marilyn Rivera
2	Baker, Andrew	Andrew Baker
3	Wilson, Aaron	Aaron Wilson
4	Rush, Samantha	Samantha Rush
5	Hutchinson, Brittany	Brittany Hutchinson
6	Abbott, Angela	Angela Abbott
7	Lambert, Alyssa	Alyssa Lambert
8	Casey, James	James Casey
9	Owens, John	John Owens
10	Cross, Brandon	Brandon Cross
11	Hernandez, Maurice	Maurice Hernandez
12	Barajas, Katherine	Katherine Barajas
13	Maldonado, Wayne	Wayne Maldonado
14	Jones, Angela	Angela Jones
15	Larson, Christina	Christina Larson
16	Wu, Fong	Fong Wu
17	Patil, Sunish	Sunish Patil
18	Joram, Koko	Koko Joram

Example: Capture Buffers for Substitution Using the PRXCHANGE Function

Suppose you have the data set Certadv.Email with email addresses, longitude, and latitude of those who have visited the company website. You are asked to reorder the longitude and latitude values to latitude and longitude.

When specifying a substitution value, you might need to rearrange pieces of the found pattern. This is possible using capture buffers.

In an earlier section, parentheses were used to represent grouping. When you use parentheses for grouping, you are creating capture buffers. Each capture buffer is referenced with a sequential number starting at 1. The first set of parentheses is for capture buffer 1. The second set of parentheses is for capture buffer 2, and so on.

When referencing a capture buffer, use a dollar sign in front of the capture buffer number. In the following example, specify the third buffer first and the first buffer last.

```
data work.latlong;
  set certadv.email;
  LatLong=prxchange('s/(-?\d+\.\d*) (@) (-?\d+\.\d*)/$3$2$1/', -1, LongLat);
run;
proc print data=work.latlong;
run;
```


Output 14.11 PROC PRINT Output of Work.LatLong (partial output)

Obs	LongLat	Email	LatLong
1	65.2874@50.8984	nichole42@hotmail.com	50.8984@65.2874
2	3.5495@115.2165	fgomez@gmail.com	115.2165@3.5495
3	88.0188@95.1651	jennifer92@gmail.com	95.1651@88.0188
4	56.2354@76.1265	rdeleon@yahoo.com	76.1265@56.2354
5	72.8874@51.1568	blee@rojas.com	51.1568@72.8874
6	23.5249@64.1968	tross@clark.org	64.1968@23.5249
7	50.1589@129.1596	williamsaaron@gmail.com	129.1596@50.1589
8	26.2291@109.0581	tammymorrow@gmail.com	109.0581@26.2291
9	2.1916@13.0526	deannadavid@gmail.com	13.0526@2.1916
10	38.5944@170.5497	daniel79@young.com	170.5497@38.5944

Quiz

- If a substring is not found in a string, the FIND function returns which value?
 - 0
 - 1
 - 2
 - Not found.
- Given the following DATA step, what is the value of USNum and WordNum?


```
data work.Count;
    Text='AUSTRALIA, ENGLAND, CANADA, AUSTRIA, ITALY, US, SPAIN';
    USNum=count(Text, 'CANADA');
    WordNum=countw(Text);
run;
```

 - 0, 7
 - 1, 7
 - 3, 0
 - 3, 0
- Which program would correctly generate two separate lagged variables for each observation?
 - ```
data work.samp1;
 set work.lag0;
 y=lag1-lag2(item);
run;
proc print data=work.samp1;
run;
```
  - ```
data work.samp2;
    set work.lag0;
```

```

        y=lag2(item);
run;
proc print data=work.samp2;
run;

c.    data work.samp3;
        set work.lag0;
        x=lag1(item);
        y=lag2(item);
run;
proc print data=work.samp3;
run;

d.    data work.samp4;
        set work.lag0;
        y=lag1(item);
        y=lag2(item);
run;
proc print data=work.samp4;
run;

```

4. What is the value of the column Position?

```
Position=prxmatch('/Dutch/', 'Sawyer Dutch Kenai');
```

- a. 2
- b. 7
- c. 8
- d. 12

5. Which program correctly searches a string for a substring and returns the position of a substring?

```

a.    data _null_;
        position=prxmatch('/mind/', 'Learning never exhausts the mind.');
```

```

        put position;
run;

b.    data _null_;
        position=prxchange('/mind/', 'Learning never exhausts the mind.');
```

```

        put position;
run;

c.    data _null_;
        position=prxparse('/mind/', 'Learning never exhausts the mind.');
```

```

        put position;
run;

d.    data _null_;
        position=findw('/mind/', 'Learning never exhausts the mind.');
```

```

        put position;
run;

```

6. Which program correctly changes the order of first and last names?

```

a.    data work.reverse;
        set certadv.reversedNames;
        name=prxmatch('/name/', name);
run;

b.    data work.reverse;

```

```

        set certadv.reversedNames;
        name=prxchange('s/(\w+),(\w+)/$2 $1/', -1, name);
run;

```

```

c.    data work.reverse;
        if _N_=1 then do;
            pattern='/name';
            name=prxparse(pattern);
        end;
        set certadv.reversedNames;
run;

```

d. None of the above.

7. Perl regular expressions in the PRXMATCH function must start and end with a delimiter.

a. True

b. False

8. Which Perl regular expression replaces the string ABC with the string ABC87?

a. 'r/ABC/ABC87/'

b. 'r/ABC87/ABC/'

c. 's/ABC87/ABC/'

d. 's/ABC/ABC87/'