Abstract
SAS® software boasts countless functions, algorithms, procedures, options, methods, code constructs, and other features to help users automate and deploy solutions for specific tasks and problems, as well as to access, transform, analyze, and manage data. This paper identifies and shares essential programming techniques that a pragmatic user and programmer should learn. Topics include determining the number of by-group levels that exist within classification variables; data manipulation with the family of CAT functions; merging or joining multiple tables of data; performing table lookup operations with user-defined formats; creating single-value and value-list macro variables with PROC SQL; examining and processing the contents of value-list macro variables; determining the FIRST., LAST. and Between by-group rows; processing repetitive data with arrays; and using metadata to better understand the contents of SAS datasets.

Introduction
As SAS users around the world celebrate a milestone of more than 40-years using SAS software, users should learn as many essential programming techniques as possible to enhance their careers well into the 21st century and beyond. Whether you're a beginner who’s just started out learning how to use SAS, an intermediate or an advanced user who has developed code, programs and/or applications using SAS software, you should do everything possible to expand your skillset. The good news is that there is no shortage of ways to learn SAS including the many SAS communities, such as communities.sas.com, blogs.sas.com, www.lexjansen.com, and many others. From new techniques to new technologies, most will help you immensely as you continue to pursue your learning objectives.

Essential Programming Techniques Every SAS User Should Learn
The most common response to the question, “What essential programming techniques should SAS users learn?” varies depending on who you ask. The fact is if you were to ask ten different SAS users what essential programming techniques a SAS user should learn you’d most likely receive a variety of responses. For some, essential programming techniques include arrays, faster programming constructs and table lookups. For others, essential programming techniques include modernizing outdated, statements, functions, options, coding constructs, algorithms and other techniques with newer, faster and more scalable programming techniques. So, what essential SAS programming techniques should be learned? In an attempt to shed some light on this very important question, I have shared a few topics, below.

Conditional Logic Scenarios
A powerful and necessary programming technique in the SAS® software is its ability to perform different actions depending on whether a programmer-specified condition evaluates to true or false. The method for accomplishing this is to use one or more conditional statements, expressions, and constructs to build a level of intelligence in a program or application. Conditional logic scenarios in the DATA step are frequently implemented using IF-THEN / ELSE and SELECT statements. The SQL procedure also supports logic scenarios and is implemented with a coding technique known as a CASE expression.

Conditional Logic with IF-THEN / ELSE
The IF-THEN / ELSE construct in the DATA step enables a sequence of conditions to be assigned that when executed proceeds through the sequence of logic conditions until a match in an expression is found or until all conditions are exhausted. The example shows a character variable Movie_Length being assigned a value of either “Shorter Length”, “Average Length”, or “Longer Length” based on the mutually exclusive conditions specified in the IF-THEN and ELSE conditions. Although not required, an ELSE condition serves as an effective technique for continuing processing to the next specified condition when a match is not found. An ELSE condition can also be useful as a “catch-all” to prevent a missing value from being assigned.
**IF-THEN / ELSE Code:**

```
LIBNAME MYDATA “E:/WORKSHOPS/WORKSHOP DATA” ;
DATA WORK.IF_THEN_EXAMPLE ;
   ATTRIB Movie_Length LENGTH=$14 LABEL='Movie Length’ ;
   SET MYDATA.MOVIES ;
   IF LENGTH < 120 THEN Movie_Length = ‘Shorter Length’ ;
   ELSE IF LENGTH > 160 THEN Movie_Length = ‘Longer Length’ ;
   ELSE Movie_Length = ‘Average Length’ ;
RUN ;
PROC PRINT DATA=WORK.IF_THEN_EXAMPLE NOOBS ;
   VAR TITLE LENGTH Movie_Length ;
RUN ;
```

**IF-THEN / ELSE Results:**

```
<table>
<thead>
<tr>
<th>Title</th>
<th>Length</th>
<th>Movie_Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brave Heart</td>
<td>177</td>
<td>Longer Length</td>
</tr>
<tr>
<td>Casablanca</td>
<td>103</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>Christmas Vacation</td>
<td>97</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>Coming to America</td>
<td>118</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>Dracula</td>
<td>120</td>
<td>Average Length</td>
</tr>
<tr>
<td>Dressed to Kill</td>
<td>105</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>Forrest Gump</td>
<td>142</td>
<td>Average Length</td>
</tr>
<tr>
<td>Ghost</td>
<td>127</td>
<td>Average Length</td>
</tr>
<tr>
<td>Jaws</td>
<td>125</td>
<td>Average Length</td>
</tr>
<tr>
<td>Jurassic Park</td>
<td>127</td>
<td>Average Length</td>
</tr>
<tr>
<td>Lethal Weapon</td>
<td>110</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>Michael</td>
<td>108</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>National Lampoon’s Vacation</td>
<td>98</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>Poltergeist</td>
<td>115</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>Rocky</td>
<td>120</td>
<td>Average Length</td>
</tr>
<tr>
<td>Scarface</td>
<td>170</td>
<td>Longer Length</td>
</tr>
<tr>
<td>Silence of the Lambs</td>
<td>118</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>Star Wars</td>
<td>124</td>
<td>Average Length</td>
</tr>
<tr>
<td>The Hunt for Red October</td>
<td>135</td>
<td>Average Length</td>
</tr>
<tr>
<td>The Terminator</td>
<td>108</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>The Wizard of Oz</td>
<td>101</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>Titanic</td>
<td>194</td>
<td>Longer Length</td>
</tr>
</tbody>
</table>
```

**Conditional Logic with SELECT-WHEN / OTHERWISE**

Another form of conditional logic available to users is a `SELECT` statement. Its purpose is to enable a sequence of logic conditions to be constructed in a DATA step by specifying one or more `WHEN` conditions and an optional `OTHERWISE` condition. When executed, processing continues through each `WHEN` condition until a match is found that satisfies the specified expression. Typically one or more `WHEN` conditions are specified in descending frequency order representing a series of conditions. The next example shows a value based on the mutually exclusive conditions specified in the sequence of logic conditions of “Shorter Length”, “Average Length”, or “Longer Length” being assigned to the character variable Movie_Length. Although not required, the `OTHERWISE` condition can be useful in the assignment of a specific value or as a “catch-all” to prevent a missing value from being assigned.
### SELECT-WHEN / OTHERWISE Code:

```
LIBNAME MYDATA “E:/WORKSHOPS/WORKSHOP DATA” ;
DATA WORK.SELECT_EXAMPLE ;
   SET MYDATA.MOVIES ;
   SELECT ;
      WHEN (LENGTH < 120) Movie_Length = ‘Shorter Length’ ;
      WHEN (LENGTH > 160) Movie_Length = ‘Longer Length’ ;
      OTHERWISE Movie_Length = ‘Average Length’ ;
   END ;
RUN ;

PROC PRINT DATA=WORK.SELECT_EXAMPLE NOOBS ;
   VAR TITLE LENGTH Movie_Length ;
RUN ;
```

### SELECT-WHEN / OTHERWISE Results:

<table>
<thead>
<tr>
<th>Title</th>
<th>Length</th>
<th>Movie_Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brave Heart</td>
<td>177</td>
<td>Longer Length</td>
</tr>
<tr>
<td>Casablanca</td>
<td>103</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>Christmas Vacation</td>
<td>97</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>Coming to America</td>
<td>118</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>Dracula</td>
<td>120</td>
<td>Average Length</td>
</tr>
<tr>
<td>Dressed to Kill</td>
<td>126</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>Forrest Gump</td>
<td>142</td>
<td>Average Length</td>
</tr>
<tr>
<td>Ghost</td>
<td>127</td>
<td>Average Length</td>
</tr>
<tr>
<td>Jaws</td>
<td>129</td>
<td>Average Length</td>
</tr>
<tr>
<td>Jurassic Park</td>
<td>137</td>
<td>Average Length</td>
</tr>
<tr>
<td>Lethal Weapon</td>
<td>110</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>Michael</td>
<td>108</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>National Lampoon’s Vacation</td>
<td>98</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>Poltergeist</td>
<td>115</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>Rocky</td>
<td>120</td>
<td>Average Length</td>
</tr>
<tr>
<td>Scareface</td>
<td>170</td>
<td>Longer Length</td>
</tr>
<tr>
<td>Silence of the Lambs</td>
<td>118</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>Star Wars</td>
<td>124</td>
<td>Average Length</td>
</tr>
<tr>
<td>The Hunt for Red October</td>
<td>135</td>
<td>Average Length</td>
</tr>
<tr>
<td>The Terminator</td>
<td>108</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>The Wizard of Oz</td>
<td>101</td>
<td>Shorter Length</td>
</tr>
<tr>
<td>Titanic</td>
<td>194</td>
<td>Longer Length</td>
</tr>
</tbody>
</table>

### Conditional Logic with CASE Expressions

Another form of conditional logic available to users is a case expression. Its purpose is to provide a way of conditionally selecting result values from each row in a table (or view). Similar to an IF-THEN/ELSE or SELECT construct in the DATA step, a case expression can only be specified in the SQL procedure. It supports a WHEN-THEN clause to conditionally process some but not all the rows in a table. An optional ELSE expression can be specified to handle an alternative action should none of the expression(s) identified in the WHEN condition(s) not be satisfied. A case expression must be a valid SQL expression and conform to syntax rules similar to DATA step SELECT-WHEN statements. Even though this topic is best explained by example, a quick look at the syntax follows.
CASE <column-name>
  WHEN when-condition THEN result-expression
  < WHEN when-condition THEN result-expression > ... 
  < ELSE result-expression >
END

A column-name can optionally be specified as part of the CASE-expression. If present, it is automatically made available to each when-condition, and is classified as a simple case expression. When it is not specified, the column-name must be coded in each when-condition, and is classified as a searched case expression. If a when-condition is satisfied by a row in a table (or view), then it is considered “true” and the result-expression following the THEN keyword is processed. The remaining WHEN conditions in the case expression are skipped. If a when-condition is “false”, the next when-condition is evaluated. SQL evaluates each when-condition until a “true” condition is found or in the event all when-conditions are “false”, it then executes the ELSE expression and assigns its value to the CASE expression’s result. A missing value is assigned to a case expression when an ELSE expression is not specified and each when-condition is “false”.

In the next example, a searched case expression is illustrated. A searched case expression in the SQL procedure provides users with the capability to perform more complex comparisons. Although the number of keystrokes can be more than with a simple case expression, the searched case expression offers the greatest flexibility and is the primary form used by SQL’ers. The noticeable absence of a column name as part of the case expression permits any number of columns to be specified from the underlying table(s) in the WHEN-THEN/ELSE logic scenarios.

The next example shows a searched case expression being used to assign the character variable Movie_Length with the AS keyword. A value of “Shorter Length” for movie lengths less than 120 minutes, “Longer Length” for movie lengths greater than 160 minutes, or “Average Length” for all other movie lengths is assigned to the newly created column. Although not required, an ELSE condition can be useful in the assignment of a specific value or as a “catch-all” to prevent a missing value from being assigned.

LIBNAME MYDATA “E:/WORKSHOPS/WORKSHOP DATA” ;
PROC SQL;
  SELECT TITLE,
         LENGTH,
         CASE
             WHEN LENGTH < 120 THEN 'Shorter Length'
             WHEN LENGTH > 160 THEN 'Longer Length'
             ELSE 'Average Length'
         END AS Movie_Length
  FROM MYDATA.MOVIES ;
QUIT ;

Searched CASE Expression Code:
As previously mentioned, searched case expressions provide users with the capability to perform more complex logic comparisons. Combined with logical and comparison operators, searched case expressions along with their WHERE clause counterparts, provide the capabilities to construct complex logic scenarios. In the next example a listing of “Action” and “Comedy” movies are displayed. Using a searched case expression, a value of “Shorter Length” for movie lengths less than 120 minutes, “Longer Length” for movie lengths greater than 160 minutes, or “Average Length” for all other movie lengths is assigned to the newly created column. A column heading of Movie_Type is assigned to the new column with the AS keyword.

```sql
LIBNAME MYDATA "E:/WORKSHOPS/WORKSHOP DATA" ;
PROC SQL;
SELECT TITLE, RATING, LENGTH, CATEGORY,
CASE
WHEN UPCASE(CATEGORY) CONTAINS 'ACTION' AND LENGTH < 120 THEN 'Action Short'
WHEN UPCASE(CATEGORY) CONTAINS 'ACTION' AND LENGTH > 160 THEN 'Action Long'
WHEN UPCASE(CATEGORY) CONTAINS 'ACTION' AND LENGTH BETWEEN 120 AND 160 THEN 'Action Medium'
WHEN UPCASE(CATEGORY) CONTAINS 'COMEDY' AND LENGTH < 120 THEN 'Comedy Short'
WHEN UPCASE(CATEGORY) CONTAINS 'COMEDY' AND LENGTH > 160 THEN 'Comedy Long'
WHEN UPCASE(CATEGORY) CONTAINS 'COMEDY' AND LENGTH BETWEEN 120 AND 160 THEN 'Comedy Medium'
ELSE 'Not Interested'
END AS MOVIE_TYPE
FROM MYDATA.MOVIES
WHERE UPCASE(CATEGORY) CONTAINS 'ACTION' OR 'COMEDY';
QUIT;
```
Gupta (2006) describes using a subsetting-IF versus a WHERE-statement or WHERE= data set option to subset observations. To avoid using a subsetting-IF statement in a DATA step, SAS users may be able to specify a WHERE= data set option for subsetting purposes directly in a procedure. This approach prevents the creation of a data set and, as a result, is more likely to scale better by reducing CPU and I/O resources. Gupta emphasizes an important detail that all SAS users should know when specifying a WHERE condition in a procedure, “Multiple WHERE conditions within SAS procedures are not cumulative as they are in a DATA step meaning the most recent WHERE condition replaces any, and all, previously specified WHERE condition(s).”

**PROC PRINT with WHERE Expression Code:**

```sas
/* WHERE Statement to Subset Observations */
proc print data=sashelp.cars noobs ;
   where type="SUV" or type="Wagon" ;
run ;
< or >

/* WHERE= Data Set Option to Subset Observations */
proc print data=sashelp.cars(where=(type="SUV" or type="Wagon")) noobs ;
run ;
```
Using the IN Operator for Comparisons

Legacy SAS applications and program code often use one, or more, OR comparison operators to handle logic scenarios. Although syntactically correct, a series of individual comparisons separated by an OR comparison operator is generally less efficient than using an IN operator. The reason is due to the way an IN operator operates. When an IN operator is specified, SAS stops making comparisons as soon as it finds a match. This is not the case with an OR operator. In the next example, a number of individual comparisons are specified using an OR operator.

**OR Comparison Operator Code:**

```
PROC SQL ;
SELECT Origin, Type, MSRP
FROM SASHELP.Cars
WHERE Type = "SUV"
   OR Type = "Truck"
   OR Type = "Wagon"
ORDER BY MSRP ;
QUIT ;
```

In the next example, an IN operator is specified to help modernize the process of handling a number of individual comparisons. The IN operator provides a convenient, and concise, way to specify scenarios with many OR comparisons. A similar example using DS2 SQLSTMT package approach can be found here.

**IN Operator Code:**

```
PROC SQL ;
SELECT Origin, Type, MSRP
FROM SASHELP.Cars
WHERE Type IN ( "SUV","Truck","Wagon" )
ORDER BY MSRP ;
QUIT ;
```

Concatenating Strings and Variables with CAT Functions

SAS functions serve an essential role in the Base SAS software. Representing a variety of built-in and callable routines, functions serve as the “work horses” in the SAS software providing users with “ready-to-use” tools designed to ease the burden of writing and testing often lengthy and complex code for a variety of programming tasks. The advantage of using SAS functions is evident by their relative ease of use, and their ability to provide a more efficient, robust and scalable approach to simplifying a process or programming task. In this example, we show how the TRIM and LEFT functions along with the concatenate operator to concatenate strings and variables together can be replaced with the CAT functions.

**CAT Function Code:**

```
data _null_;
  length NUM 3. A B C D E $ 8 BLANK $ 1 ;
  A = 'The' ;
  NUM = 5 ;
  B = ' Cats' ;
  C = 'in' ;
  D = ' the' ;
  E = 'Hat' ;
  BLANK = ' ' ;
```
Essential Programming Techniques Every SAS® User Should Learn, continued
PharmaSUG 2021

Old approach with TRIM / LEFT functions and concatenation operator:

```sas
* Old approach with TRIM / LEFT functions and concatenation operator;
OLD=trim(left(A)) || BLANK || trim(left(NUM)) || BLANK || trim(left(B)) ||
    BLANK || trim(left(C)) || BLANK || trim(left(D)) || BLANK || trim(left(E));
```

* Using the CAT functions to concatenate character and numeric values together:

1. CAT = cat (A, NUM, B, C, D, E);
2. CATQ = catq(BLANK, A, NUM, B, C, D, E);
3. CATS = cats(A, NUM, B, C, D, E);
4. CATT = catt(A, NUM, B, C, D, E);
5. CATX = catx(BLANK, A, NUM, B, C, D, E);

```sas
put OLD= / STRIP= / CAT= / CATQ= / CATS= / CATT= / CATX= / ;
r
```

**CAT Function Results:**

<table>
<thead>
<tr>
<th>OLD</th>
<th>CAT</th>
<th>CATQ</th>
<th>CATS</th>
<th>CATT</th>
<th>CATX</th>
</tr>
</thead>
<tbody>
<tr>
<td>The 5 Cats in the Hat</td>
<td>The 5 Cats in the Hat</td>
<td>&quot;The 5 Cats in the Hat&quot;</td>
<td>The 5 Cats in the Hat</td>
<td>The 5 Cats in the Hat</td>
<td>The 5 Cats in the Hat</td>
</tr>
</tbody>
</table>

**Analysis:**

In the example, above, a single numeric variable, NUM, and six character variables: A, B, C, D, E, and BLANK are defined with their respective values as: NUM=5, A='The', B=' Cats', C='in', D=' the', E='Hat' and BLANK=' '. The oldest way of concatenating two or more strings or variables together is then specified, using the TRIM and LEFT functions with the concatenation operator "||" in an assignment statement. As an alternative, a newer and more robust concatenation approach is specified using the CAT family of functions: CAT, CATQ, CATS, CATT, and CATX.

1. **CAT**, the simplest of concatenation functions, joins two or more strings and/or variables together, end-to-end producing the same results as with the concatenation (double bar) operator.

2. **CATQ** is similar to the default features of the CATX function, but the CATQ function adds quotation marks to any concatenated string or variable.

3. **CATS** removes leading and trailing blanks and concatenates two or more strings and/or variables together.

4. **CATT** removes trailing blanks and concatenates two or more strings and/or variables together.

5. **CATX**, perhaps the most robust CAT function, removes leading and trailing blanks and concatenates two or more strings and/or variables together with a user-specified delimiter between each.

**Concatenating (or Appending) Data Sets**

Concatenating data sets is the process of combining two, or more, data sets, one after the other, with the purpose of creating a single data set. The number of observations in the new data set is the sum total of observations in all the original input data sets. The order of observations in the concatenated data set is arranged sequentially with the observations from the first data set, followed by the observations from the second data set, and so on. The concatenated data set contains the same variables as the input data sets. Should an input data set contain different variables from the other input data sets, the concatenated data set will have missing values assigned to the variables from the other input data sets.
Concatenating with the DATA-SET-RUN Construct
SAS provides users with a few ways to concatenate data sets. In the first example, below, an old-style DATA-SET construct is specified to concatenate the two data sets, RUGs_2015 and RUGs_2016. Although syntactically correct, this approach does not scale well because it forces SAS to incur heavy I/O (input/output) because the observations in each input data set must be read and written to the concatenated data set.

**DATA-SET-RUN Code:**

```sas
data WORK.Concatenated_Results ;
  set WORK.RUGs_2015 WORK.RUGs_2016 ;
run ;
```

**DATA-SET-RUN Results:**

<table>
<thead>
<tr>
<th>RUG</th>
<th>Number_Papers</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>MWsug</td>
<td>96</td>
<td>2015</td>
</tr>
<tr>
<td>SCSug</td>
<td>29</td>
<td>2015</td>
</tr>
<tr>
<td>SEsug</td>
<td>145</td>
<td>2015</td>
</tr>
<tr>
<td>Wuss</td>
<td>102</td>
<td>2015</td>
</tr>
<tr>
<td>MWsug</td>
<td>124</td>
<td>2016</td>
</tr>
<tr>
<td>SCSug</td>
<td>62</td>
<td>2016</td>
</tr>
<tr>
<td>SEsug</td>
<td>148</td>
<td>2016</td>
</tr>
<tr>
<td>Wuss</td>
<td>112</td>
<td>2016</td>
</tr>
</tbody>
</table>

Concatenating with a PROC SQL Outer Union CORR
A second approach uses PROC SQL to concatenate data sets. In this next example, an OUTER UNION CORR set operator is specified, and SQL reads and processes the tables in each query producing a new concatenated table of results.

**PROC SQL Code:**

```sas
proc sql ;
  create table WORK.Concatenated_Results as
  select * from WORK.RUGs_2015
  outer union corr
  select * from WORK.RUGs_2016 ;
  select * from WORK.Concatenated_Results ;
quit ;
```

**PROC SQL Results:**

<table>
<thead>
<tr>
<th>RUG</th>
<th>Number_Papers</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>MWsug</td>
<td>96</td>
<td>2015</td>
</tr>
<tr>
<td>SCSug</td>
<td>29</td>
<td>2015</td>
</tr>
<tr>
<td>SEsug</td>
<td>145</td>
<td>2015</td>
</tr>
<tr>
<td>Wuss</td>
<td>102</td>
<td>2015</td>
</tr>
<tr>
<td>MWsug</td>
<td>124</td>
<td>2016</td>
</tr>
<tr>
<td>SCSug</td>
<td>62</td>
<td>2016</td>
</tr>
<tr>
<td>SEsug</td>
<td>148</td>
<td>2016</td>
</tr>
<tr>
<td>Wuss</td>
<td>112</td>
<td>2016</td>
</tr>
</tbody>
</table>
Concatenating with PROC APPEND (or PROC DATASETS – APPEND Statement)

A third, and more efficient, concatenation approach is available to SAS users. Using PROC APPEND (or the APPEND statement in PROC DATASETS), an input data set can be appended to another data set. The advantage of using this approach is reduced I/O, since SAS does not have to read the observations in the base data set. Appending this way offers a way to scale an application. As the number of observations in the base data set grows, the advantage of using this approach can become huge. In the next example, two PROC APPENDs are specified to concatenate the observations in the RUGs_2015 and RUGs_2016 data sets.

**PROC APPEND Code:**
```sas
proc append base=WORK.Concatenated_Results
   Data=WORK.RUGs_2015;
run;
proc append base=WORK.Concatenated_Results
   Data=WORK.RUGs_2016;
run;
```

**PROC APPEND Results:**

<table>
<thead>
<tr>
<th>RUG</th>
<th>Number_Papers</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVSGU</td>
<td>95</td>
<td>2015</td>
</tr>
<tr>
<td>SCSUG</td>
<td>25</td>
<td>2015</td>
</tr>
<tr>
<td>SESUG</td>
<td>148</td>
<td>2015</td>
</tr>
<tr>
<td>WISS</td>
<td>102</td>
<td>2015</td>
</tr>
<tr>
<td>MVSGU</td>
<td>124</td>
<td>2016</td>
</tr>
<tr>
<td>SCSUG</td>
<td>62</td>
<td>2016</td>
</tr>
<tr>
<td>SESUG</td>
<td>148</td>
<td>2016</td>
</tr>
<tr>
<td>WISS</td>
<td>112</td>
<td>2016</td>
</tr>
</tbody>
</table>

Processing Multiple TABLE Statements with PROC FREQ

Benjamin (2012) describes a common problem programmers have when using PROC FREQ to produce multiple table results. Programmers will often code two, or more, individual PROC FREQ and TABLE statements even for the same input data set. Although the PROC FREQ code, illustrated below, is syntactically correct, invoking PROC FREQ multiple times in this way can result in an increase in the amount of time for processing the request.

**PROC FREQ Code:**
```sas
proc freq data=sashelp.cars;
   table Origin / list out=work.Origin_Freq1;
run;
proc freq data=sashelp.cars;
   table Origin * Type / list out=work.Origin_Freq2;
run;
proc freq data=sashelp.cars;
   table Origin * Type * Cylinders / list out=work.Origin_Freq3;
run;
```

To optimize the code, programmers can force a single pass over the input data set and as a result reduce the amount of processing time needed to produce the resulting data sets, as follows.
Essential Programming Techniques Every SAS® User Should Learn, continued

Optimized PROC FREQ Code:

```sas
proc freq data=sashelp.cars;
  table Origin / list out=work.Origin_Freq1;
  table Origin * Type / list out=work.Origin_Freq2;
  table Origin * Type * Cylinders / list out=work.Origin_Freq3;
run;
```

List of Procedures Supporting a CLASS Statement

Procedures are classified as the “workhorses” in the SAS System. The CLASS statement specifies one, or more, character or numeric variables used to group data into classification levels. A virtue of using a CLASS statement is that a SORT procedure is not required to arrange and group the data, because the stats and other information is collected in memory and reported at the end of the procedure. A partial list of SAS procedures, below, supports the use of a CLASS statement.

<table>
<thead>
<tr>
<th>SAS Procedures Supporting a CLASS Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC ANOVA</td>
</tr>
<tr>
<td>PROC DISCRIM</td>
</tr>
<tr>
<td>PROC GENMOD</td>
</tr>
<tr>
<td>PROC GLM</td>
</tr>
<tr>
<td>PROC LOGISTIC</td>
</tr>
</tbody>
</table>

Producing Page Numbers with ODS RTF Pagination Functions

Page numbering is the process of applying a sequence of numbers, Roman numerals, or letters on reports, spreadsheets, documents, books or other multi-page files. Legacy applications and program code frequently use counters or code routines to generate and display page numbers. Simple page numbering routines may resemble something similar to the following code.

DATA Step Code:

```sas
FILENAME REPORT DISK 'c:\DATA_NULL_Report.LST';
DATA _NULL_;
  SET SASHHELP.CARS END=EOF;
  FILE REPORT HEADER=H1; /* Execute Page_Header Routine */
  PUT @1 Origin $6.  
    @10 Make $13.  
    @25 MSRP DOLLAR12. ;
  RETURN ;
H1: ; /* Page Header */
  Page_CTR + 1 ;
  PUT @15 DATA _NULL_ Detail Report
    // @22 'Page Number ' Page_CTR ;
  RETURN ;
RUN ;
```
Page numbers can be produced and displayed in RTF output by specifying an escape character with an ODS RTF statement, any of the following functions, and an ODS RTF CLOSE ; statement:

- \{thispage\}
- \{lastpage\}
- \{pageof\}

Page Counters with ODS RTF Functions

Output Delivery System (ODS) provides powerful features that users can use when producing output. In the next example, an escape character is specified with the ODS RTF destination, where the functions: \{thispage\}, \{lastpage\}, and \{pageof\} are specified in the title and footnote statements to produce the page numbers and the total number of pages in the report.

```
ods escapechar='^';
ods RTF file='c:\Print-Report.rtf';
proc print data=sashelp.cars noobs;
  title 'Page \{thispage\} of \{lastpage\}';
  footnote '\{pageof\}';
run;
ods RTF close;
```

### ODS RTF Code:

```
ods escapechar='^';
ods RTF file='c:\Print-Report.rtf';
proc print data=sashelp.cars noobs;
  title 'Page \{thispage\} of \{lastpage\}';
  footnote '\{pageof\}';
run;
ods RTF close;
```
**Automating the Process of Creating Multiple HTML Files**

The Web offers incredible potential that impacts all corners of society. With its increasing popularity as a communications medium, Web publishers have arguably established the Web as the greatest medium ever created. Businesses, government agencies, professional associations, schools, libraries, research agencies, and a potpourri of society's true believers have endorsed the Web as an efficient means of conveying their messages to the world.

The SAS software provides users with the capability to create results and deploy selected pieces of output as HTML output files. Using the Output Delivery System (ODS) HTML destination, output can be created that anyone can view using a web browser. Syntactically correct HTML code is automatically produced and made ready for deployment using one of the Internet browser software products (e.g., Internet Explorer, Google Chrome, Mozilla Firefox, Safari, etc.). As a result, the SAS System and the HTML destination create a type of “streaming” or continuous output by adding elevator bars (horizontal and/or vertical) for easy navigation.

In the following example, redundant code and hardcoding issues are avoided by using PROC SQL to determine the number of unique (or distinct) values of the Origin column exist and once known are assigned to single-value and value-list macro variables. With the unique values assigned to two macro variables, an iterative %DO statement is specified to control the propagation of one, or more, HTML files containing one-way frequency results. The results of the three distinct HTML files that were created are also displayed, below.

```
/* Output HTML Files Location */
filename odsout "E:\" ;

options symbolgen ;
%macro multfiles ;
proc sql noprint ;
select count(distinct origin)
into :morigin_cnt  /* derive number of origins */
from sashelp.cars
order by origin ;
select distinct origin
into :morigin_list separated by "~" /* derive unique origin values */
from sashelp.cars
order by origin ;
quit ;

%do i=1 %to &morigin_cnt ;
ods html path=odsout (URL=None)
      file="%SCAN(&morigin_list,&i,~)_FrequencyReport (MultiHTMLFiles).html"
      style=styles.barrettsblue ;
title "Cars with Origin in %SCAN(&morigin_list,&i,~)" ;
proc freq data=sashelp.cars(where=(origin = "%SCAN(&morigin_list,&i,~)") ;
tables type ;
      format msrp dollar12.0 ;
run ;
quit ;
title ;
ods html close ;
%end ;
%put &morigin_list ;
%mend multfiles ;

%multfiles ;
```
Automating the Process of Creating Multiple Excel Files

Statistics show that the world's most used software application is Microsoft Excel®. Due to this dominance, SAS provides users with several ways to send results, tables, statistics, images and other output directly to an Excel spreadsheet. In the next example, redundant code and hardcoding issues are avoided by using PROC SQL to determine the number of unique (or distinct) values of the Origin column and, once known, are assigned to single-value and value-list macro variables. With the values assigned to the two macro variables, an iterative %DO statement is specified to control the propagation of Excel files containing one-way frequency results. The results of the three distinct Excel files that were created are also displayed, below.

ODS Excel Code:

```sas
%macro multExcelfiles ;
proc sql noprint ;
    select count(distinct origin)
        into :morigin_cnt /* derive number of origins */
        from sashelp.cars
        order by origin ;
    select distinct origin
        into :morigin_list separated by "~" /* derive unique origin values */
        from sashelp.cars
        order by origin ;
quit ;

%do i=1 %to &morigin_cnt ;
    ods Excel file="e:\%SCAN(&morigin_list,&i,~)_FreqReport (MultiExcelFiles).xlsx"
        style=styles.barrettsblue ;
    title "Cars with Origin in %SCAN(&morigin_list,&i,~)" ;
    proc freq data=sashelp.cars(where=(origin = "%SCAN(&morigin_list,&i,~)") ) ;
        tables type ;
        format msrp dollar12.0 ;
    run ;
    quit ;
    title ;
    ods Excel close ;
%end ;
%put &morigin_list ;
%mend multExcelfiles ;
%multExcelfiles ;
```
Discovering the Number of Occurrences of a Value in a Data Set

Discovering the number of occurrences of individual values in a data set is useful information, particularly when constructing data-driven approaches. SAS provides several ways to count and determine the number of occurrences of a value in a data set.

Discovering the Number of Occurrences of a Value in a DATA Step

One approach for discovering the number of occurrences of a variable’s value(s) is to construct a DATA step counting routine. In the next example, individual counters for the number of females and males are created, and after the last observation is read and processed, the results for each counter is output to the Counts data set, and the results displayed with PROC PRINT.

```sas
data WORK.Counts(drop=Sex) ;
set sashelp.Heart(keep=Sex) end=EOF ;
  if Sex = "Female" then Number_Females + 1 ;
  else if Sex = "Male" then Number_Males + 1 ;
  if EOF then do ;
    Total = Number_Females + Number_Males ;
    format Number_Females Number_Males Total comma7. ;
    output ;
  end ;
run ;
proc print data=WORK.Counts noobs ;
run ;
```

Discovering the Number of Occurrences of a Value with the PROC FREQ NLEVELS Option

Another approach for counting the number of occurrences of a variable’s value(s) is to specify the NLEVELS option in PROC FREQ. In this example, the variable SEX is kept and the NLEVELS option is specified for the SASHELP.Heart data set. The results show there are two levels for the variable, SEX, with 2,873 females and 2,336 males.

```sas
proc freq data=sashelp.Heart(keep=sex) NLEVELS ;
run ;
```
Discovering the Number of Occurrences of a Value with PROC SQL
Another approach for counting the number of occurrences of a variable’s value is to use the SUM function with an equality expression in PROC SQL. PROC SQL’s data access and query capabilities offer SAS users with a powerful approach to summing down rows and across columns. In this example, a SELECT query is specified with a SUM function for counting the number of “Females”, “Males” and their combined totals that are found in the SASHELP.HEART data set. An optional FORMAT=COMMA7. parameter is also specified to make the results easier to read.

Using Metadata to Determine the Number of Observations in a Data Set
Metadata is everywhere and is defined as information that describes data. Other definitions include information about data, or information about the design and specification of objects and data structures. In its most basic form, metadata is found in the cataloging systems of every academic library, public library, school library, and special library in the world. The typical book, magazine, microfiche, digital file, image, or object’s metadata is stored in cataloging systems. These cataloging systems are not composed of words, sentences, paragraphs, or chapters, but contain information about its author(s), title, subject, keyword(s), description, publisher, publication date, ISBN, format, resource identifier, copyright, and other information.

Older Methods of Determining the Number of Observations in a Data Set
Before the availability of metadata in the SAS System, users developed and included code routines that determined the number of observations in a data set. An often used DATA step approach, since the beginning of SAS-time, constructs a variable that counts the number of observations. Although syntactically correct, this approach does not “scale” well – due to the amount of I/O incurred and the sizes of data sets – when computing the counter. The the next example, a DATA step approach computes the total number of “Sedans” found in the SASHELP.CARS data set, and displays the results using PROC PRINT.
DATA Step Code:

data WORK.sedans_counter(keep=type obs_ctr)
   WORK.cars_sedans(drop=obs_ctr);
set sashelp.cars(keep=origin type make MSRP) end=eof;
   where upcase(type) = "SEDAN";
   obs_ctr + 1;
output cars_sedans;
   if eof then output sedans_counter;
run;

proc print data=WORK.sedans_counter noobs;
run;

DATA Step Results:

<table>
<thead>
<tr>
<th>Type</th>
<th>obs_ctr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sedan</td>
<td>262</td>
</tr>
</tbody>
</table>

Using DICTIONARY.TABLES Metadata to Determine the Number of Observations in a Data Set

The SAS System collects and populates valuable metadata about SAS libraries, data sets (tables), catalogs, indexes, macros, system options, titles, views and other useful information in a collection of read-only tables called Dictionary tables. Dictionary tables serve a special purpose for SAS users by providing system-related information about the current SAS session’s SAS databases and applications. When a query processes a Dictionary table, SAS automatically launches a discovery process at runtime to collect information pertinent to that table. This information is made available any time after a SAS session is started.

When users need more information about SAS data sets the TABLES Dictionary table can be very helpful. The TABLES Dictionary table provides detailed information about the library names, the member (or data set) names, the date a data set was created and last modified, the number of observations in a data set, and much more. The next example illustrates a popular approach that accesses the metadata content from the DICTIONARY.TABLES table to determine the number of observations in any SAS data set.

PROC SQL Code:

title "Number of Rows in a Table";

proc sql;
   select libname, memname, nobs format=comma10.
   from Dictionary.Tables
   where nobs NE 0;
quit;
title;
Using SASHELP.VTABLE Metadata to Determine the Number of Observations in a Data Set

SAS also provides users with metadata content in a number of SASHELP views. In this next example the number of observations in any SAS data set can be determined by accessing the NOBS metadata content in the SASHELP.VTABLE view. This metadata content can be displayed using any output-producing SAS procedure, as shown below.

```sas
title "Number of Rows in a Table" ;
proc print data=sashelp.vtable noobs ;
   var libname memname nobs ;
   format nobs comma10. ;
   where nobs NE . ;
run ;
title ;
```
Using PROC PRINT with Style

Hecht (2011) describes the appearance of PROC PRINT output can be customized with colors, backgrounds, fonts, justifications, and other report components using styles. Styles can be specified for all destinations (e.g., RTF, PDF, HTML, Excel, etc.) except the Listing destination. In the next example, the SASHELP.CARS data set is sorted in ascending order by the variables Origin, Type, Make and MSRP; the HTML destination is opened with the HTMLBlue style selected for output; and background and foreground styles selected for the data, obs and total parts of the PROC PRINT report output.

PROC PRINT Code:

```sas
proc sort data=sashelp.CARS(keep=Origin Type Make MSRP)
   out=WORK.Cars_Sorted ;
   where MSRP < 20000 ;
   by Origin Type Make MSRP ;
run ;
ods HTML path="/folders/myfolders" (url=none)
   file="PROC-PRINT-with-Style.html"
   style=HTMLBlue ;
proc print data=WORK.Cars_Sorted
   style (data) = [background=Blue foreground=white]
   style (obs)  = [background=red foreground=white]
   style (total) = [background=yellow foreground=black] ;
   by Origin Type ;
```

**PROC PRINT Results:**

<table>
<thead>
<tr>
<th>item</th>
<th>itemname</th>
<th>nobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORK</td>
<td>COUNTS</td>
<td>1</td>
</tr>
<tr>
<td>MYDATA</td>
<td>ACTORS</td>
<td>13</td>
</tr>
<tr>
<td>MYDATA</td>
<td>ACTORES_WITH_MESSY_DATA</td>
<td>15</td>
</tr>
<tr>
<td>MYDATA</td>
<td>AE</td>
<td>13</td>
</tr>
<tr>
<td>MYDATA</td>
<td>DM</td>
<td>24</td>
</tr>
<tr>
<td>MYDATA</td>
<td>EX</td>
<td>146</td>
</tr>
<tr>
<td>MYDATA</td>
<td>MOVIES</td>
<td>22</td>
</tr>
<tr>
<td>MYDATA</td>
<td>MOVES_WITH_MESSY_DATA</td>
<td>31</td>
</tr>
<tr>
<td>SASHELP</td>
<td>ACACMP</td>
<td>2,020</td>
</tr>
<tr>
<td>SASHELP</td>
<td>AARMN</td>
<td>100</td>
</tr>
<tr>
<td>SASHELP</td>
<td>ADISSC</td>
<td>126</td>
</tr>
<tr>
<td>SASHELP</td>
<td>APPSNG</td>
<td>1,080</td>
</tr>
<tr>
<td>SASHELP</td>
<td>AR</td>
<td>144</td>
</tr>
<tr>
<td>SASHELP</td>
<td>ARPAURO</td>
<td>156</td>
</tr>
<tr>
<td>SASHELP</td>
<td>ASSCMGR</td>
<td>403</td>
</tr>
<tr>
<td>SASHELP</td>
<td>BASEBALL</td>
<td>222</td>
</tr>
<tr>
<td>SASHELP</td>
<td>BBD</td>
<td>24,200</td>
</tr>
<tr>
<td>SASHELP</td>
<td>BMNMIN</td>
<td>1,284</td>
</tr>
<tr>
<td>SASHELP</td>
<td>BMT</td>
<td>137</td>
</tr>
<tr>
<td>SASHELP</td>
<td>BROWNS</td>
<td>24,591</td>
</tr>
<tr>
<td>SASHELP</td>
<td>BUY</td>
<td>11</td>
</tr>
<tr>
<td>SASHELP</td>
<td>BWIGHT</td>
<td>50,000</td>
</tr>
<tr>
<td>SASHELP</td>
<td>CARS</td>
<td>428</td>
</tr>
<tr>
<td>SASHELP</td>
<td>CITDAY</td>
<td>1,069</td>
</tr>
<tr>
<td>SASHELP</td>
<td>CITHON</td>
<td>145</td>
</tr>
<tr>
<td>SASHELP</td>
<td>CITIOTR</td>
<td>48</td>
</tr>
<tr>
<td>SASHELP</td>
<td>CIWINK</td>
<td>319</td>
</tr>
<tr>
<td>SASHELP</td>
<td>CITIYR</td>
<td>10</td>
</tr>
<tr>
<td>SASHELP</td>
<td>CLASSG</td>
<td>19</td>
</tr>
<tr>
<td>SASHELP</td>
<td>CLASPT</td>
<td>19</td>
</tr>
</tbody>
</table>

SASHELP | QUAKE     | 15,576 |
SASHELP | RENT      | 10    |
SASHELP | METAL     | 56    |
SASHELP | PHFU42    | 72    |
SASHELP | ROCKPIT   | 6     |
SASHELP | SASMC     | 0     |
SASHELP | SASMS9    | 784   |
SASHELP | SHOES     | 395   |
SASHELP | SUJUMK    | 1,703  |
SASHELP | SMEMSG    | 94    |
SASHELP | SPRNGS    | 1,587  |
SASHELP | STEEL     | 44    |
SASHELP | STOCKS    | 699   |
SASHELP | STTMSG    | 336   |
SASHELP | SVRTDIST  | 2,373  |
SASHELP | SYM1001   | 105   |
SASHELP | TABLE     | 6     |
SASHELP | THXK      | 15    |
SASHELP | TIMEDATA  | 40,330 |
SASHELP | TOURISM   | 49    |
SASHELP | USECON    | 232   |
SASHELP | VEBPLAYRS | 11    |
SASHELP | VEBSMGR   | 19    |
SASHELP | VEBTSG    | 9     |
SASHELP | VICSIG    | 7     |
SASHELP | VOTE1980  | 3,197  |
SASHELP | WEBNIG    | 345   |
SASHELP | WORKERS   | 67    |
SASHELP | YR1001    | 125   |
SASHELP | YR111     | 126   |
SASHELP | ZHC       | 7,445  |
SASHELP | ZIPBODE   | 41,232 |
SASHELP | ZTC       | 15,181 |
SASHELP | _OMPIRK_   | 44    |
Using Available Memory with Hash Object Programming

Dorfman (2009) describes a SAS hash object as, “a high-performance look-up table residing completely in the DATA step memory.” Due to the costs and availability of memory resources in today’s computing environments, software vendors are doing everything they can to develop language constructs that capitalize on memory-resident operations. Dorfman further describes that, “The hash object is implemented via a Data Step Component Interface (DSCI), meaning that it is not a part of the DATA step proper. Rather, picture it as a black-box device you can manipulate from inside the DATA step to ask it for lightning-quick data storage and retrieval services.”

Lafler (2016) describes a SAS hash object as, “a data structure that contains an array of items that are used to map identifying values, known as keys (e.g., employee IDs), to their associated values (e.g., employee names or employee addresses). As implemented, a hash object in the SAS System is used as a DATA step construct and is not available to any SAS Procedures.” A hash object reads the contents of a data set into memory once allowing the SAS system to repeatedly access the data, as necessary. The contents of a hash object can be saved to a SAS data set (or table), but at the end of the DATA step the hash object will be dropped.

```sas
proc print results;
    id Origin Type Make;
    format msrp dollar12.0;
    sum MSRP;
    run;
    ods HTML close;
```

### PROC PRINT Results:

<table>
<thead>
<tr>
<th>Origin</th>
<th>Type</th>
<th>Make</th>
<th>MSRP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asia</td>
<td>Hybrid</td>
<td>Honda</td>
<td>$19,110</td>
</tr>
<tr>
<td></td>
<td>SUV</td>
<td>Honda</td>
<td>$10,600</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Kia</td>
<td>$16,056</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mitsubishi</td>
<td>$16,662</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Suzuki</td>
<td>$17,163</td>
</tr>
<tr>
<td></td>
<td>SUV</td>
<td>$34,240</td>
<td></td>
</tr>
<tr>
<td>Asia</td>
<td>Sports</td>
<td>Hyundai</td>
<td>$18,730</td>
</tr>
<tr>
<td></td>
<td>Truck</td>
<td>Mazda</td>
<td>$14,040</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Nissan</td>
<td>$19,479</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Toyota</td>
<td>$12,500</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Toyota</td>
<td>$18,405</td>
</tr>
<tr>
<td></td>
<td>Truck</td>
<td>$63,014</td>
<td></td>
</tr>
<tr>
<td>Asia</td>
<td>Wagon</td>
<td>Kia</td>
<td>$11,906</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mitsubishi</td>
<td>$17,495</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Suzuki</td>
<td>$19,487</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Toyota</td>
<td>$16,668</td>
</tr>
<tr>
<td></td>
<td>Wagon</td>
<td>$75,767</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wagon</td>
<td>$974,425</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Origin</th>
<th>Type</th>
<th>Make</th>
<th>MSRP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Europe</td>
<td>Sedan</td>
<td>MINI</td>
<td>$10,969</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MINI</td>
<td>$19,999</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Volkswagen</td>
<td>$18,715</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Volkswagen</td>
<td>$19,826</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$70,538</td>
<td></td>
</tr>
<tr>
<td>Europe</td>
<td>Wagon</td>
<td>Volkswagen</td>
<td>$19,000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$34,453</td>
<td></td>
</tr>
<tr>
<td>Origin</td>
<td>Type</td>
<td>Make</td>
<td>MSRP</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>USA</td>
<td>Sports</td>
<td>Ford</td>
<td>$18,345</td>
</tr>
<tr>
<td></td>
<td>Truck</td>
<td>Dodge</td>
<td>$17,830</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ford</td>
<td>$14,385</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GMC</td>
<td>$16,530</td>
</tr>
<tr>
<td></td>
<td>Truck</td>
<td>$67,305</td>
<td></td>
</tr>
<tr>
<td>Origin</td>
<td>Type</td>
<td>Make</td>
<td>MSRP</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>USA</td>
<td>Wagon</td>
<td>Ford</td>
<td>$17,476</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pontiac</td>
<td>$17,045</td>
</tr>
<tr>
<td></td>
<td>Wagon</td>
<td>$34,520</td>
<td></td>
</tr>
<tr>
<td>USA</td>
<td></td>
<td>$495,400</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1,564,372</td>
<td></td>
</tr>
</tbody>
</table>


object and all its contents disappear. Since memory-based operations are typically faster than their disk-based counterparts, users often experience faster and more efficient table lookup, merge, sort and transpose operations.

Users with DATA step programming experience will find the hash object syntax relatively straightforward to learn and use. Available in all operating systems running SAS 9 or greater, the hash object is called using methods. The syntax for calling a method involves specifying the name of the user-assigned hash table, a dot (.), the desired method (e.g., operation) by name, and finally the specification for the method enclosed in parentheses. The following example illustrates the basic syntax for calling a method to define a key.

```
MatchTitles.DefineKey ('Title');
```

where MatchTitles is the name of the hash table, DefineKey is the name of the called method, and ‘Title’ is the specification being passed to the method.

An essential operation frequently performed by users is the process of table lookup or search. The hash object as implemented in the DATA step provides users with the necessary tools to conduct match-merges (or joins) of two or more data sets. Data does not have to be sorted (or be in a designated sort order) before use as it does with the DATA step merge process. The next example illustrates a hash object with a simple key (TITLE) to merge (or join) the MOVIES and ACTORS data sets to create a new data set (MATCH_ON_MOVIE_TITLES) with matched observations.

```
data match_on_movie_titles(drop=rc) ;
  if 0 then set mydata.movies
      mydata.actors ; /* load variable properties into hash tables */
  if _n_ = 1 then do ;
    declare Hash MatchTitles (dataset:'mydata.actors') ; /* declare the name MatchTitles for hash */
    MatchTitles.DefineKey ('Title') ; /* identify variable to use as key */
    MatchTitles.DefineData ('Actor_Leading',
      'Actor_Supporting') ; /* identify columns of data */
    MatchTitles.DefineDone () ; /* complete hash table definition */
  end ;
  set mydata.movies ;
  if MatchTitles.find(key:title) = 0 then output ; /* lookup TITLE in MOVIES table using MatchTitles hash */
run ;
```

Hash Object Code:
Conclusion
As SAS users around the world celebrate a milestone of more than 40-years using SAS software, users should learn as many essential programming techniques as possible to enhance their careers well into the 21st century and beyond. This paper identified and shared several essential programming techniques that a pragmatic user and programmer should learn. From topics related to conditional logic scenarios; subsetting with WHERE expressions; determining the number of by-group levels that exist within classification variables; data manipulation with the family of CAT functions; merging or joining multiple tables of data; performing table lookup operations with user-defined formats; creating single-value and value-list macro variables with PROC SQL; examining and processing the contents of value-list macro variables; determining the FIRST., LAST. and Between by-group rows; processing repetitive data with arrays; and using metadata to better understand the contents of SAS datasets.

References

Efficiency and Performance Tuning References and Suggested Reading

Hash Object References and Suggested Reading

**Macro References and Suggested Reading**


**SAS Programming Techniques References and Suggested Reading**


Acknowledgments
The author thanks the 2021 PharmaSUG Conference Committee for accepting my abstract and paper; the 2021 PharmaSUG Executive Committee for organizing and supporting this virtual conference event; SAS Institute Inc. for providing SAS users with wonderful software; and SAS users everywhere for being the nicest people anywhere!

Trademark Citations
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.

Data Sets Used in Examples
The examples presented in this paper include the RUGs_2015 and RUGs_2016 data sets; and several in the SASHELP library including the CARS, and HEART data sets. You’ll be able to access and use these data sets for example purposes and for testing the enclosed code examples.

The RUGs_2015 data set consists of 4 observations and 3 variables, illustrated below.

![Data Set #1. RUGs_2015](image1)

The RUGs_2016 data set consists of 4 observations and 3 variables, illustrated below.

![Data Set #2. RUGs_2016](image2)

The MOVIES data set consists of 22 observations and 6 variables, illustrated below.
The ACTORS data set consists of 13 observations and 3 variables, illustrated below.

The SASHELP.CARS data set consists of 428 observations and 15 variables, illustrated below.

The SASHELP.HEART data set consists of 5,209 observations and 17 variables, illustrated below.
### Author Bio

Kirk Paul Lafler is a lecturer and adjunct professor at San Diego State University; an advisor and adjunct professor at the University of California San Diego Extension; and teaches SAS, SQL, Python, R and Excel courses, seminars, workshops, and webinars to students, professionals and users around the world. Kirk has been a SAS user since 1979 and is the author of several books including, PROC SQL: Beyond the Basics Using SAS, Third Edition (SAS Press. 2019) along with papers and articles on a variety of SAS topics. Kirk has also been selected as an invited speaker, educator, keynote and section leader at SAS conferences; and is the recipient of 25 “Best” contributed paper, hands-on workshop (HOW), and poster awards.

Comments and suggestions can be sent to:

Kirk Paul Lafler  
SAS® Consultant, Application Developer, Programmer, Data Analyst, Educator and Author  
E-mail: KirkLafler@cs.com  
LinkedIn: [https://www.linkedin.com/in/KirkPaulLafler/](https://www.linkedin.com/in/KirkPaulLafler/)  
LinkedIn: [https://www.linkedin.com/in/Order-of-Magnitude-Analytics/](https://www.linkedin.com/in/Order-of-Magnitude-Analytics/)  
Twitter: @sasNerd

---

**Data Set #6. SASHELP.HEART**

<table>
<thead>
<tr>
<th>Status</th>
<th>Death/Cause</th>
<th>Age/Char</th>
<th>Sex</th>
<th>Age/Start</th>
<th>Height</th>
<th>Weight</th>
<th>Waistline</th>
<th>Systolic</th>
<th>Diastolic</th>
<th>BMI</th>
<th>Smoking</th>
<th>Age/Death</th>
<th>Occupation</th>
<th>Marital Status</th>
<th>BP Status</th>
<th>Weight_Status</th>
<th>Smoking_Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dead</td>
<td>Other</td>
<td>Female</td>
<td>25</td>
<td>62.5</td>
<td>140</td>
<td>78</td>
<td>70</td>
<td>124</td>
<td>121</td>
<td>0</td>
<td>55</td>
<td>Dead</td>
<td>Low</td>
<td>Single</td>
<td>Normal</td>
<td>overweight</td>
<td>Non-smoker</td>
</tr>
<tr>
<td>2</td>
<td>Cancer</td>
<td>Female</td>
<td>57</td>
<td>62.25</td>
<td>132</td>
<td>50</td>
<td>172</td>
<td>114</td>
<td>110</td>
<td>0</td>
<td>200</td>
<td>High</td>
<td>Low</td>
<td>Single</td>
<td>Normal</td>
<td>overweight</td>
<td>Non-smoker</td>
</tr>
<tr>
<td>4</td>
<td>Alive</td>
<td>Female</td>
<td>35</td>
<td>65.75</td>
<td>108</td>
<td>80</td>
<td>128</td>
<td>123</td>
<td>0</td>
<td>0</td>
<td>240</td>
<td>Normal</td>
<td>Normal</td>
<td>Married</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>5</td>
<td>Alive</td>
<td>Male</td>
<td>42</td>
<td>66</td>
<td>156</td>
<td>76</td>
<td>110</td>
<td>116</td>
<td>20</td>
<td>0</td>
<td>281</td>
<td>Optimal</td>
<td>Normal</td>
<td>Married</td>
<td>Normal</td>
<td>overweight</td>
<td>Heavy (25-29)</td>
</tr>
<tr>
<td>6</td>
<td>Alive</td>
<td>Female</td>
<td>58</td>
<td>61.75</td>
<td>131</td>
<td>52</td>
<td>175</td>
<td>117</td>
<td>0</td>
<td>0</td>
<td>196</td>
<td>Desirable</td>
<td>Normal</td>
<td>Single</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>7</td>
<td>Alive</td>
<td>Female</td>
<td>36</td>
<td>64.75</td>
<td>136</td>
<td>80</td>
<td>132</td>
<td>110</td>
<td>15</td>
<td>0</td>
<td>196</td>
<td>Desirable</td>
<td>Normal</td>
<td>Married</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>8</td>
<td>Dead</td>
<td>Male</td>
<td>53</td>
<td>65.5</td>
<td>130</td>
<td>80</td>
<td>114</td>
<td>99</td>
<td>0</td>
<td>77</td>
<td>276</td>
<td>High</td>
<td>Normal</td>
<td>Married</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>9</td>
<td>Alive</td>
<td>Male</td>
<td>35</td>
<td>71</td>
<td>194</td>
<td>60</td>
<td>132</td>
<td>124</td>
<td>0</td>
<td>0</td>
<td>211</td>
<td>Optimal</td>
<td>Normal</td>
<td>Married</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>10</td>
<td>Dead</td>
<td>Male</td>
<td>52</td>
<td>62.5</td>
<td>129</td>
<td>79</td>
<td>124</td>
<td>106</td>
<td>6</td>
<td>3</td>
<td>234</td>
<td>Desirable</td>
<td>High</td>
<td>Single</td>
<td>Normal</td>
<td>overweight</td>
<td>Heavy (25-29)</td>
</tr>
<tr>
<td>11</td>
<td>Alive</td>
<td>Female</td>
<td>59</td>
<td>66.25</td>
<td>175</td>
<td>76</td>
<td>123</td>
<td>133</td>
<td>30</td>
<td>0</td>
<td>235</td>
<td>Normal</td>
<td>Very Heavy</td>
<td>Married</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>12</td>
<td>Alive</td>
<td>Male</td>
<td>33</td>
<td>64.25</td>
<td>191</td>
<td>60</td>
<td>108</td>
<td>110</td>
<td>0</td>
<td>0</td>
<td>221</td>
<td>Optimal</td>
<td>Normal</td>
<td>Married</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>13</td>
<td>Alive</td>
<td>Male</td>
<td>55</td>
<td>70</td>
<td>174</td>
<td>75</td>
<td>142</td>
<td>114</td>
<td>0</td>
<td>0</td>
<td>188</td>
<td>Desirable</td>
<td>Normal</td>
<td>Married</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>14</td>
<td>Alive</td>
<td>Male</td>
<td>57</td>
<td>67.25</td>
<td>105</td>
<td>76</td>
<td>128</td>
<td>110</td>
<td>15</td>
<td>0</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
<td>Married</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>15</td>
<td>Alive</td>
<td>Male</td>
<td>66</td>
<td>69</td>
<td>195</td>
<td>90</td>
<td>130</td>
<td>105</td>
<td>30</td>
<td>0</td>
<td>232</td>
<td>Desirable</td>
<td>Normal</td>
<td>Married</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>16</td>
<td>Alive</td>
<td>Female</td>
<td>37</td>
<td>64.5</td>
<td>134</td>
<td>75</td>
<td>120</td>
<td>100</td>
<td>10</td>
<td>0</td>
<td>196</td>
<td>Desirable</td>
<td>Normal</td>
<td>Single</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>17</td>
<td>Alive</td>
<td>Male</td>
<td>40</td>
<td>66.25</td>
<td>151</td>
<td>72</td>
<td>152</td>
<td>112</td>
<td>30</td>
<td>0</td>
<td>152</td>
<td>Desirable</td>
<td>Very Heavy</td>
<td>Married</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>18</td>
<td>Dead</td>
<td>Male</td>
<td>56</td>
<td>67.25</td>
<td>122</td>
<td>72</td>
<td>120</td>
<td>87</td>
<td>15</td>
<td>72</td>
<td>154</td>
<td>Desirable</td>
<td>Underweight</td>
<td>Married</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>19</td>
<td>Alive</td>
<td>Female</td>
<td>42</td>
<td>67.75</td>
<td>162</td>
<td>56</td>
<td>138</td>
<td>119</td>
<td>1</td>
<td>0</td>
<td>269</td>
<td>Desirable</td>
<td>Normal</td>
<td>Single</td>
<td>Light (1-10)</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>20</td>
<td>Dead</td>
<td>Male</td>
<td>46</td>
<td>66.5</td>
<td>157</td>
<td>84</td>
<td>142</td>
<td>116</td>
<td>30</td>
<td>76</td>
<td>233</td>
<td>Desirable</td>
<td>Very Heavy</td>
<td>Married</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>21</td>
<td>Alive</td>
<td>Female</td>
<td>37</td>
<td>66.25</td>
<td>160</td>
<td>78</td>
<td>110</td>
<td>112</td>
<td>15</td>
<td>0</td>
<td>192</td>
<td>Desirable</td>
<td>Normal</td>
<td>Married</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>22</td>
<td>Alive</td>
<td>Female</td>
<td>45</td>
<td>64</td>
<td>167</td>
<td>74</td>
<td>120</td>
<td>119</td>
<td>5</td>
<td>0</td>
<td>209</td>
<td>Desirable</td>
<td>Normal</td>
<td>Married</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>23</td>
<td>Alive</td>
<td>Female</td>
<td>59</td>
<td>65.75</td>
<td>156</td>
<td>74</td>
<td>156</td>
<td>122</td>
<td>0</td>
<td>0</td>
<td>209</td>
<td>Desirable</td>
<td>Normal</td>
<td>Married</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>24</td>
<td>Alive</td>
<td>Female</td>
<td>36</td>
<td>63.75</td>
<td>122</td>
<td>84</td>
<td>132</td>
<td>132</td>
<td>0</td>
<td>0</td>
<td>184</td>
<td>Desirable</td>
<td>Normal</td>
<td>Married</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>25</td>
<td>Alive</td>
<td>Female</td>
<td>50</td>
<td>61.5</td>
<td>105</td>
<td>60</td>
<td>150</td>
<td>106</td>
<td>15</td>
<td>0</td>
<td>228</td>
<td>Desirable</td>
<td>Normal</td>
<td>Married</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
</tbody>
</table>