Paper 105-2023

10 Quick Tips for Getting Tipsy with SAS®

Lisa Mendez, Catalyst Clinical Research;
Richann Jean Watson, DataRich Consulting

ABSTRACT

There are many useful tips that do not warrant a full paper but put some cool SAS® code together and you get a cocktail of SAS code goodness. This paper will provide ten great coding techniques that will help enhance your SAS programs. We will show you how to 1) add quotes around each tokens in a text string, 2) create column headers based values using Transpose Procedure (PROC TRANSPOSE), 3) set missing values to zero without using a bunch of if-then-else statements, 4) using short-hand techniques for assigning lengths of consecutive variables and for initializing to missing, 5) calculating the difference between the visit's actual study day and the target study and accounting for no day zero (0); 6) use SQL Procedure (PROC SQL) to read variables from a data set into a macro variable, 7) use the XLSX engine to read data from multiple Microsoft® Excel® Worksheets, 8) test to see if a file exists before trying to open it, such as an Microsoft Excel file, 9) using the DIV function to divide so you don't have to check if the value is zero, and 10) use abbreviations for frequently used SAS code.

INTRODUCTION

While there are other papers available that describe some of the functions that are used in this paper, this paper is meant to provide a quick reference to coding tips that utilize the functions that can help enhance your SAS code. It is not meant to be an in-depth dive into any one concept, but rather provides you with techniques that you can use to solve various coding problems, or to provide efficiency. These tips will help you to get started coding and enhance your existing code. Many times, you don't know what you don't know, and you find ways to solve problems, but they may not necessarily be the most "elegant" ways. By adding these 10 coding tips to your coding toolbox, you will save time and effort.

TIP 1: ADDING QUOTES AROUND EACH TOKEN IN A STRING

How many times have you been given a list of character values that need to be used to subset your data? Normally, the lists are provided without each individual item enclosed in quotations. To use the list in an SAS statement these values need to be quoted. We could manually add the quotation marks to each item especially if it is a static list, but what if you a list that could change (i.e., data-driven list). How do we add those quotes?

In SAS Program 1, we show how we can add quotations to each item in our list. We utilize several SAS pre-defined functions. Because we are not interested in saving the list in a data set, we use DATA _NULL_ and create a temporary variable that will capture the new quoted list. First, we determine how many tokens we have with the COUNTW function. We then loop through each item adding quotations using the QUOTE function. After we have looped through each item, we then use CALL SYMPUTX subroutine to save the temporary variable to a macro variable. Data Display 1 shows our two macro variables, the original one we started with and the newly quoted macro variable.

```
%let fmtname = SUPPDMQNAM SUPPCMQNAM;
data _null_;
   fmtname = "&fmtname";
   length newfmtname $500;
   do i = 1 to countw(fmtname, ' ');
      newfmtname = catx(' ', newfmtname, quote(scan(fmtname, i)));
   end;
   call symputx('newfmtname', newfmtname);
run;
```

SAS Program 1: Adding Quotes to Individual Tokens

fmtname	newfmtname
SUPPDMQNAM SUPPCMQNAM	"SUPPDMQNAM" "SUPPCMQNAM"

Data Display 1: Original Macro Variable and Newly Quoted Macro Variable

TIP 2: COLUMN HEADERS WITH PROC TRANSPOSE

As we work to create outputs for various deliverables, we need to consider the incorporation of population counts in the column headers. Most of the time this can be done using macros and while that is easy enough to do, it can be tedious if you have several outputs that have the same overlying structure, but the column headers may differ. Of course, these can be handled using the typical technique of macro variables, but this requires you to know in advance the different populations and how many columns are needed for each output. This could make it a bit cumbersome to reuse code; however, there is a way to make this data driven.

In order to illustrate the concept of using a data-driven approach to create column headers with the population counts, we need to create a sample data set. In our sample data set, we have the different treatments (TRTP), and the population counts (PCNT) for each of the treatments. We can combine these into one variable to create a treatment label (TRTPLBL) as seen in Data Display 2. In addition, we have a numeric variable for each treatment. The numeric variable ideally should be sequential and represent the order the data is to be presented. This comes into play in Tip 3. AVAL represents the counts per treatment group that is to be displayed in the report. It is only used for illustration purposes and has no significant meaning. Note the special character (`) in the label. This special character comes into play when producing the output.

TRTPN	TRTP	PCNT	TRTPLBL	AVAL
1	Drug A	90	Drug A`(N=90)	86
2	Drug A + Comp C	90	Drug A + Comp C`(N=90)	75
3	Drug B	150	Drug B`(N=150)	137
4	Drug B + Comp D	90	Drug B + Comp D`(N=90)	.
5	Drug A Overall	180	Drug A Overall`(N=180)	161
6	Drug B Overall	240	Drug B Overall`(N=240)	137

Data Display 2: Sample Data Set for Using Labels as Column Headers

Using our sample data set, we use PROC TRANSPOSE to convert the rows to columns (SAS Program 2). With the use of the ID statement, we indicate that TRTPN is to represent the column names. The PREFIX option on the PROC TRANSPOSE statements, indicate that the variable names are to start with 'TRT_'. We then use the IDLABEL statement to assign a label to each of the variables created using the ID statement. Data Display 3 shows the transposed data set. However, it is not readily evident that the labels have been assigned to each of the variable names.

```
proc transpose data = trt
               out = t_trt (drop = _:)
               prefix = TRT_;
   var AVAL;
   id TRTPN;
   idlabel TRTPLBL;
run;
```

SAS Program 2: Transpose Data to Convert from Rows to Columns

TRT_1	TRT_2	TRT_3	TRT_4	TRT_5	TRT_6
86	75	137	.	161	137

Data Display 3: Sample Data Set Transposed

Using the CONTENTS Procedure, we see that the labels have been assigned to each variable in Data Display 4

Name	Type	Length	Format	Informat	Label
TRT_1	Numeric	8			Drug A`(N=90)
TRT_2	Numeric	8			Drug A + Comp C`(N=90)
TRT_3	Numeric	8			Drug B`(N=150)
TRT_4	Numeric	8			Drug B + Comp D`(N=90)
TRT_5	Numeric	8			Drug A Overall`(N=180)
TRT_6	Numeric	8			Drug B Overall`(N=240)

Data Display 4: Contents of the Transposed Data Set

Once the data are in the appropriate format, we can use the REPORT Procedure to produce the output. Notice that in PROC REPORT, if all the variables are to be displayed, and they are in the order in which you want them displayed, then we can use _ALL_ on the COLUMN statement. In addition, it is not necessary to specify the DEFINE statements since by default PROC REPORT will use the labels associated with the variable. Of course, if you need to control other aspects of the columns, such as individual column widths, the data type or order, then a DEFINE statement is needed. But for illustration purposes, we used a simple PROC REPORT (SAS Program 3) to produce TIP2.RTF (Data Display 5). The special character that was embedded in the variable labels is used on the PROC REPORT statement to indicate that this is where we want the label to split when writing to the ODS destination.

```
options orientation = landscape leftmargin = 1in rightmargin = 1in nodate;
ods rtf file = "C:\Users\gonza\Desktop\Conferences\Drafts\Tipsy\TIP2.rtf";
proc report data = t_trt split = "`"
            style(column) = [just = c cellwidth = 1.35in];
   columns _all_;
run;
ods rtf close;
run;
```

SAS Program 3: PROC REPORT to Produce Output

Drug A (N=90)	Drug A + Comp C (N=90)	Drug B (N=150)	Drug B + Comp D (N=90)	Drug A Overall (N=180)	Drug B Overall (N=240)
86	75	137	.	161	137

Data Display 5: TIP2.RTF

TIP 3: MISSING VALUES DEFAULT TO ZERO

Now that we have our output, we see that there is a column with a missing value. Again, for a static set of variables, this could be easily defaulted to zero with IF-THEN-ELSE. In this example, we would need 6 statements. But if in another output we only had 3 treatments or another had 8 treatments, the code would need to be adjusted accordingly. A dynamic approach allows the population of 0 for null values without using a bunch of IF-THEN-ELSE statements. Utilizing the ARRAY statement, we can loop through each variable in the transposed data set. For this specific technique, the array name is the prefix "TRT_" associated with the variables that are created using PROC TRANSPOSE illustrated in Tip 2. The subscript indicated in the ARRAY statement is a macro variable that contains the number of treatments that can be determined from the data. The COALESCEC function is used to return the first non-missing value in the list specified, (i.e., TRT_{i} or 0). If TRT_{i} is null, then the 0 is returned. SAS Program 4 is used to produce Data Display 6.

```
data t_trt2 (drop = i);
   set t_trt;
   array TRT_{&num_trt} 3;
   do i = 1 to dim(TRT_);
      TRT_{i} = coalesce(TRT_{i}, 0);
   end;
run;
```

SAS Program 4: Setting Null Values to Zero

TRT_1	TRT_2	TRT_3	TRT_4	TRT_5	TRT_6
86	75	137	0	161	137

Data Display 6: Revised Data Set with Null Values Set to Zero

TIP 4: SHORTHAND FOR DEALING WITH CONSECUTIVE VARIABLES AND DATA SETS WITH SIMILAR NAMES

This next tip shows you a shorthand way to set a bunch of variables to missing as well how to set a bunch of data sets together that have a similar name.

SHORTHAND FOR INITIALIZING VARIABLES TO MISSING

Typically, when defaulting variables to missing, we use an assignment statement for each variable as seen in SAS Program 5. This approach requires you to use the correct syntax based on the data type, i.e., a period for numeric or a set of quotes for character. But before we initialize the variables to missing, we need to assign a length to each variable so that the character variables are not length of one and the numeric variables do not default to a length of 8.

```
data temp;
   length varnmc1 varnmc2 varnmc3 varnmc4 varnmc5 varnmc6 varnmc7
          varnmc8 varnmc9 varnmc10 $200
          varnmn1 varnmn2 varnmn3 varnmn4 varnmn5 varnmn6 varnmn7
          varnmn8 varnmn9 varnmn10 varnmn11 varnmn12 varnmn13
          varnmn14 varnmn15 3.;
   varnmc1 = '';
   varnmc2 = '';
   …
   varnmc10 = '';
   varnmn1 = .;
   varnmn2 = .;
   …
   varnmn15 = .;
run;
```

SAS Program 5: Assigning Missing Values on Individual Assignment Statements

There is a much easier way to produce the same results as shown in SAS Program 5. Since all the character variables were numbered consecutively and have the same length and all the numeric variables are also numbered consecutively with the same length, we can utilize the *var1 - varN* approach to assign the lengths of the variables. We then use the CALL MISSING subroutine to initialize all the variables. SAS Program 6 demonstrates this technique, with SAS Program 7 showing two alternatives for an even more abbreviated technique. With SAS Program 7, we take advantage of the wildcard character," : ", to indicate any variable that has the prefix will be part of the list of variables to be initialized.

```
/* shorthand for assigning lengths of consecutive vars and for initializing to missing */
data temp;
   length varnmc1 - varnmc10 $200 varnmn1 - varnmn15 3.;
   call missing(of varnmc1 - varnmc10, of varnmn1 - varnmn15);
run;
```

SAS Program 6: Assigning Missing Values Using a Shorthand Technique

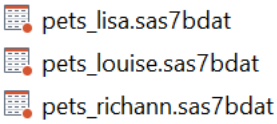
```
data temp2;
   length varnmc1 - varnmc10 $200 varnmn1 - varnmn15 3.;
   call missing(of varnmc:, of varnmn:);
run;

data temp3;
   length varnmc1 - varnmc10 $200 varnmn1 - varnmn15 3.;
   call missing(of varnm:);
run;
```

SAS Program 7: Alternatives to SAS Program 6

SHORTHAND FOR SETTTING DATA SETS WITH THE SAME PREFIX

There are times when we need to pull data from several different sources and set some indicator variable to identify the source. If we specify each data set individually, we can use the IN = option on the data set along with IF-THEN-ELSE statements to set this indicator variable. In the example, there are three data sets by owners (Data Display 7). SAS Program 8 illustrates how these can be combined into a master file with a new variable, OWNER, added.



Data Display 7: List of Data Sets that Need to be Combined.

```
libname TIPSY "C:\Users\gonza\Desktop\Conferences\Drafts\Tipsy";
data allpetsorig;
   set TIPSY.PETS_LISA (in = lisa)
       TIPSY.PETS_LOUISE (in = louise)
       TIPSY.PETS_RICHANN (in = richann);
   length OWNER $20;
   if lisa then OWNER = 'Lisa';
   else if louise then OWNER = 'Louise';
   else if richann then OWNER = 'Richann';
run;
```

SAS Program 8: Combining All the Pets Data into a Master File

If the data sets have the same prefix, a shorthand approach could be implemented using the colon wildcard operator and the INDSNAME option. The wildcard operator indicates that we are to pull all data sets in the indicated library (e.g., TIPSY) that have the same root name (e.g., PETS_). The INDSNAME allows us to identify the source data set which can be used to create the OWNER variable. Keep in mind that INDSNAME will provide the full name of the data set including the library name. For example, TIPSY.PETS_LISA, TIPSY.PETS_LOUISE and TIPSY.PETS_RICHANN is what is returned in SOURCE. Since we only want the owners name, we can use several SAS functions to retrieve only the portion of the data set that contains the name (see SAS Program 9).

```
libname TIPSY "C:\Users\gonza\Desktop\Conferences\Drafts\Tipsy";
data allpets;
   set TIPSY.PETS_:
       indsname = source;
   length OWNER $20;
   OWNER = propcase(scan(source, -1, '_'));
run;
```

SAS Program 9: Combining All the Pets Data into a Master File Using a Shorthand Technique

Utilizing this shorthand approach allows us to add more data to the master PETS data set without having to modify the code.

```
53          data allpets;
54             set TIPSY.PETS_:
55                 indsname = source;
56             length OWNER $20;
57             OWNER = propcase(scan(source, -1, '_'));
58          run;

NOTE: There were 5 observations read from the data set TIPSY.PETS_LISA.
NOTE: There were 6 observations read from the data set TIPSY.PETS_LOUISE.
NOTE: There were 5 observations read from the data set TIPSY.PETS_RICHANN.
NOTE: The data set WORK.ALLPETS has 16 observations and 15 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

SAS Log 1: Log from SAS Program 9

Note that this technique also works with the MERGE statement if all the data sets have the same key variables that are used for merging (SAS Program 10 and SAS Log 2).

```
data richann_pets;
   merge TIPSY.RJW_PET:;
   by name;
run;
```

SAS Program 10: Merging All Pet Information into a Master Data

```
29          data richann_pets;
30             merge TIPSY.RJW_PET:;
31             by name;
32          run;

NOTE: There were 5 observations read from the data set TIPSY.RJW_PETBASIC.
NOTE: There were 5 observations read from the data set TIPSY.RJW_PETDEMO.
NOTE: There were 5 observations read from the data set TIPSY.RJW_PETOTHSP.
NOTE: The data set WORK.RICHANN_PETS has 5 observations and 13 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

SAS Log 2: Log from SAS Program 10

TIP 5: ABSOLUTE DIFFERENCE BETWEEN STUDY DAY AND TARGET DAY

Within CDISC ADaM standards variables ending in DY represent the relative days and therefore there is no Day 0. Thus, when calculating DY variables, we implement programming logic to determine if the reference date is on or after the analysis date to see if we need to add 1. Similarly, since there is no Day 0, special care needs to be taken when using the analysis visit window variable, AWTDIFF (Analysis Window Diff from Target). AWTDIFF represents the absolute difference between the target day (AWTARGET) and the relative day (ADY). SAS Program 11 shows how we can account for no Day 0 when calculating ADY and AWTDIFF.

```
data dt_dy;
   set dts;
   if nmiss(ADT, TRTSDT) = 0 then ADY = ADT - TRTSDT + (ADT >= TRTSDT);
   if nmiss(ADY, AWTARGET) = 0 then AWTDIFF = abs(ADY - AWTARGET) - ((ADY*AWTARGET) < 0);
run;
```

SAS Program 11: Accounting for No Day 0 When Calculating ADY and AWTDIFF

Notice in Data Display 8 that if we did not account for no Day 0, then for the record with ADT = 31OCT2022, the absolute difference from the target date would be off by one day. Which could make a difference between selecting one record over another for analysis purposes.

TRTSDT	ADT	AWTARGET	ADY	AWTDIFF	AWTDIFF *
01NOV2022	29OCT2022	1	-3	3	4
01NOV2022	31OCT2022	1	-2	2	3
01NOV2022	01NOV2022	1	1	0	0
01NOV2022	14DEC2022	42	44	2	2
01NOV2022	30JAN2023	84	91	7	7

* Not adjusted for no Day 0

Data Display 8: Calculating ADY and AWTDIFF When There is No Day 0

TIP 6: MACRO VARIABLE LIST USING PROC SQL

How many times have you created a macro and then invoked the macro by a list of macro statements? This "hard coding" technique yields a static list of macro calls which would require updating as the more elements are added or elements are removed. But what if you could drive the macro calls based on values from a column in a data set? This data-driven technique is more efficient, because if the data values change (maybe an addition or exclusion of a data element) the list is updated automatically and by extension the macro calls are updated.

This tip shows how you can concatenate the values of one column of a data set into one macro variable separated by a delimiter. This is useful for building up a list of variables or constants based on values in a data set. You can also use the SQLOBS macro variable to reveal how many distinct variables there are in the data processed by the query.

In Data Display 9, we see three variables from the SASHELP.BASEBALL data set: NAME, TEAM, and POSITION. Let's create a macro variable that will be a string of distinct variables separated by an asterisk (*).

Player's Name	Team at the End of 1986	Position(s) in 1986
Allanson, Andy	Cleveland	C
Ashby, Alan	Houston	C
Davis, Alan	Seattle	1B
Dawson, Andre	Montreal	RF
Galarraga, Andres	Montreal	1B
Griffin, Alfredo	Oakland	SS

Data Display 9: SASHELP.BASEBALL Partial Data Set

Using PROC SQL, we will create a macro variable with a string of unique values of the POSITION variable separated by an asterisk. We use SQLOBS to obtain the number of variables in the string and assign it to a macro variable named N_VARS. We then write out the string and the N_VARS macro variable in the log so we can see the values and the distinct number of variables in the string (SAS Program 12).

```
proc sql noprint;
       select distinct Position
       into :Position_List separated by '*'
       from sashelp.baseball;
       %let n_vars = &SQLOBS;
quit;

%put &=n_vars &=position_list ;  /* print variable list in the log */
```

SAS Program 12: Create New Data Set with Unique Values of Variable Position & Print String in Log

Since we used the “noprint” option in our PROC SQL code, the results of the query are not written to the results window. In addition, since there is no CREATE TABLE statement, a physical data set is not written out to the WORK library but is held in memory and stored in a macro variable named POSITION_LIST. We can see what values are stored in the macro variable by writing the macro variable to the log using the %put statement. And an added tip is to add an equal sign (=) in between the ampersand (&) and the macro variable name to display the macro variable name in the log. SAS Log 3 shows the unique variables in the macro variable, which are separated by the asterisk character.

```
     %put &n_vars &position_list ;  /* print variable list in the log */
N_VARS=25
POSITION_LIST=13*1B*1O*23*2B*2S*32*3B*3O*3S*C*CD*CF*CS*DH*DO*LF*O1*OD*OF*OS*RF*S3*SS*UT
```

SAS Log 3: Log from SAS Program 12

To invoke the macro variable, you will need to use the scan function, which will scan the macro variable string and select text between the asterisks. In the do loop, you can use the number of variables, or use a data-driven variable to get the count.

In SAS Program 13, we create a data set for each baseball position. We will sort the baseball data set before invoking the macro.

```
/* sort the data set by position */
proc sort data = baseball;
       by position;
run;

/* Create seperate datasets for each position */
%macro position_datasets;
       %do i = 1 %to &n_vars;
       %let var = %scan(&Position_List,&i,*);

       data Baseball_&var;
              set sashelp.baseball;
              where position = "&var";
       run;

       %end; /* of do loop */

%mend; /* end macro position_datasets */

/* invoke macro */
%position_datasets;
```

SAS Program 13: Invoke Macro Using the Scan Function to Obtain Macro Variable

After the program runs, you should have 25 new data sets in the WORK library, each one with observations pertaining to only one position (see Figure 1).

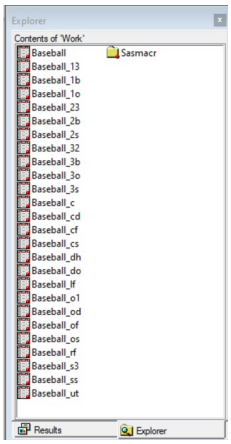


Figure 1: Output Data Sets in WORK Library.

TIP 7: USING THE XLSX ENGINE TO READ IN MULTIPLE EXCEL WORKSHEETS

Did you know that the SAS XLSX engine allows you to read and write Microsoft Excel files as if they were data sets in a library? The advantage is that it accesses the XLSX file directly; it does not use the Microsoft data APIs as a go-between. The catch is, you must have a license for SAS/Access to PC Files. (Hemedinger, 2015)

Let's say we have an Excel workbook that has three worksheets: English, Math, and TX History, and we want to create three data sets for these worksheets.

We start by using the LIBNAME statement that uses the XLSX engine:

```
libname Grades XLSX "C:\Users\lamen\OneDrive\Documents\!Personal\PharmaSUG 2023\ Grade 5
Grade Book - Mendez.xlsx";
```

SAS Program 14: LIBNAME Statement Using XLSX Engine

In the LIBNAME statement, you point to where the Excel workbook is located. When you run the statement, the structure of library is set up (see Figure 2).

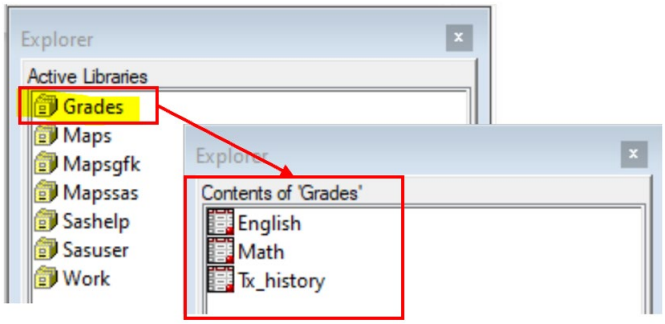


Figure 2: The Grades Library is Created by the XLSX Engine with Names of the Worksheets as Data Set Names.

Note: Because variable names and worksheet names often have spaces or characters that are not used in valid SAS naming conventions, you can use the global option "VALIDVARNAME=ANY" to allow those names to be read in by SAS. (DelGobbo, 2006)

```
options validvarname=any;
```

SAS Program 15: Optional Options Statement to Allow SAS to Read in All Variable Names

To load the data from the worksheets into the data sets, you can use the SAS Dictionary tables (Brill I., 2006) and Tip #7 to read the list of data set names into a macro variable, and then read the Excel worksheet data into the data sets. SAS Program 16provides the example code.

```
/* ------------------------------------------------------------------ */
/* Create a dataset with all the Excel Worksheet Names                */
/* (dataset names in the grades library)                              */
/* ------------------------------------------------------------------ */
proc sql noprint;
       create table grades.grades_list as
       select *
       from sashelp.vmember
       where libname = "GRADES";
quit;

/* ------------------------------------------------------------------ */
/* Create a macro variable with list of dataset names (subjects)      */
/* ------------------------------------------------------------------ */
proc sql noprint;
       select memname
       into :subject separated by '*'
       from grades.grades_list;
       %LET n_vars = &SQLOBS;
quit;

%put &n_vars &subject ;  /* print variable list in the log */

/* ------------------------------------------------------ */
/* Read the names of the worksheets, create and populates  */
/* all of the data in the worksheet into the datasets in   */
/* the library.                                            */
/* ------------------------------------------------------ */
%macro populate;
       %do i = 1 %to &n_vars;
       %let var = %scan(&subject,&i,*);

       PROC IMPORT out = grades.&var
              datafile = "C:\Users\lamen\OneDrive\Documents\!Personal\PharmaSUG 2023\Grade
5 Grade Book - Mendez.xlsx";
                     DBMS = XLSX REPLACE;
              sheet="&var";
              getnames=yes;
       RUN;

       %end; /* of do loop */

%mend; /* end macro populate */

/* invoke macro */
%populate;
```

SAS Program 16: Create a Data Set with Data Set Names from Grades Library, Create a Macro Variable with List of Data Set Names, and Macro to Read in Excel Worksheets Data into Data Sets

TIP 8: CHECK FOR FILE EXISTENCE OR SUCCESSFUL ASSIGNMENT OF A LIBNAME

How many times have you executed a program and after it finishes executing realize that it did not execute successfully because a file (or a LIBNAME) did not exist? Checking for the existence of files or successful assignment of LIBNAMEs early in the execution process can help save time. If the file does not exist or LIBNAME was not successfully assigned, then you can build in exception handling that will return a message to the log and halt execution.

SAS has built-in functions that will allow us to check the existence of a file or the successful assignment of a LIBNAME. FILEEXIST checks for the existence of a file and returns a value of 1 if the file is found and a value of 0 if the file is not found. LIBREF checks for the successful assignment of a LIBNAME. If it is successful, then it returns a value of 0, otherwise it returns a non-zero value. Both functions are invoked using %SYSFUNC. SAS Program 17 illustrates a macro that checks to ensure the file is found or

the LIBNAME is assigned before proceeding to the next step. If the condition is not met, then a message is written to the log and the macro execution is stopped. SAS Log 4 demonstrates the outcome when a file is found and when one is not found as well as when a LIBNAME is successfully assigned and not assigned.

```
libname tipsy 'C:\Users\gonza\Desktop\Conferences\Drafts\Tipsy';
libname tips2 'C:\Users\gonza\Desktop\Conferences\Drafts\Tips';

%macro lfexist(libfile = , lfcheck = libname);
   %if %upcase(&lfcheck) = LIBNAME %then %let lffunc = %sysfunc(libref(&libfile)) = 0;
   %else %let lffunc = %sysfunc(fileexist(&libfile)) = 1; ;

   %if &lffunc %then %do;
      %put &libfile WAS FOUND!;
      %put ADDITIONAL SAS CODE WOULD BE ENTERED HERE TO CONTINUE PROCESSING;
   %end;
   %else %do;
      %put %sysfunc(compress(E RROR:)) &libfile WAS NOT FOUND!!!;
      %abort;
   %end;
%mend lfexist;

%lfexist(libfile = C:\Users\gonza\Desktop\Conferences\Drafts\Tipsy\Pet Data.xlsx,
         lfcheck = file)
%lfexist(libfile = C:\Users\gonza\Desktop\Conferences\Drafts\Tipsy\Pet_Data.xlsx,
         lfcheck = file)
%lfexist(libfile = tipsy)
%lfexist(libfile = tips2)
```

SAS Program 17: Macro to Check Existence of a File or Successful Assignment of a LIBNAME

```
42         %lfexist(libfile = C:\Users\gonza\Desktop\Conferences\Drafts\Tipsy\Pet
Data.xlsx, lfcheck = file)
C:\Users\gonza\Desktop\Conferences\Drafts\Tipsy\Pet Data.xlsx WAS FOUND!
ADDITIONAL SAS CODE WOULD BE ENTERED HERE TO CONTINUE PROCESSING
43         %lfexist(libfile =
C:\Users\gonza\Desktop\Conferences\Drafts\Tipsy\Pet_Data.xlsx, lfcheck = file)
ERROR: C:\Users\gonza\Desktop\Conferences\Drafts\Tipsy\Pet_Data.xlsx WAS NOT FOUND!!!
ERROR: Execution terminated by an %ABORT statement.
44         %lfexist(libfile = tipsy)
tipsy WAS FOUND!
ADDITIONAL SAS CODE WOULD BE ENTERED HERE TO CONTINUE PROCESSING
45         %lfexist(libfile = tips2)
ERROR: tips2 WAS NOT FOUND!!!
ERROR: Execution terminated by an %ABORT statement.
```

SAS Log 4: Log for SAS Program 17

TIP 9: DIVIDING NUMBERS USING THE DIVIDE FUNCTION

A number of outputs that we generate require us to determine a percentage or a ratio. Thus, we need to determine the numerator and denominator and then decide if we can even perform the calculations. If we just divide the numerator by the denominator without ensuring the denominator is non-zero (SAS Program 18) then we could end up with a division by zero and receive an unwanted log message (SAS Log 5). While this is not a warning, it is something we do not want to see in our logs. To circumvent this log message, we would need to employ an IF-THEN-ELSE technique to verify that the denominator is not zero, before we perform the calculation.

```
data div;
   set num_den;
   DIV = NUM / DEN;
run;
```

SAS Program 18: Dividing Without Confirmation Denominator is NOT Zero

```
29          data div;
30             set num_den;
31             DIV = NUM / DEN;
32          run;

NOTE: Division by zero detected at line 31 column 14.
NUM=39 DEN=0 DIV=. _ERROR_=1 _N_=3
NOTE: Mathematical operations could not be performed at the following places. The results
of the operations have been set to missing values.
      Each place is given by: (Number of times) at (Line):(Column).
      1 at 31:14
NOTE: There were 5 observations read from the data set WORK.NUM_DEN.
NOTE: The data set WORK.DIV has 5 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

SAS Log 5: Log from SAS Program 18

Although using IF-THEN-ELSE is a valid technique, there is another technique that will allow us to perform the calculation without checking for a non-zero denominator. The DIVIDE function returns the calculated value or if it is unable to calculate it will return a special missing value. SAS Program 19 illustrates the use of the DIVIDE function. Notice that the log does not yield any messages regarding the division by zero (SAS Log 6). The calculation is performed and if there is a division by zero then it returns a special missing value, e.g., "I " for infinity (Data Display 10).

```
data div;
   set num_den;
   DIV = divide(NUM, DEN);
run;
```

SAS Program 19: Dividing Using the DIVIDE Function

```
28          data div;
29             set num_den;
30             DIV = divide(NUM, DEN);
31          run;

NOTE: There were 5 observations read from the data set WORK.NUM_DEN.
NOTE: The data set WORK.DIV has 5 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

SAS Log 6: Log from SAS Program 19

NUM	DEN	DIV	DIV*
34	3	11.33333	11.33333
125	10	12.5	12.5
39	0	.	I
0	-3	0	0
-44	-4	11	11

* Uses the DIVIDE function

Data Display 10: Data Set Using the DIVIDE Function

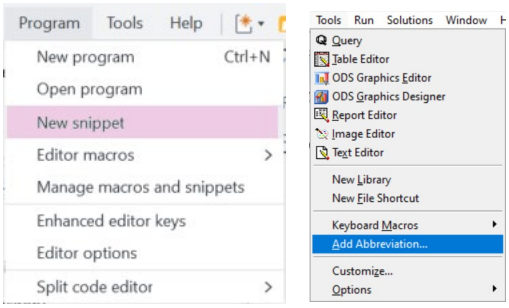
TIP 10: ABBREVIATIONS FOR FREQUENTLY USED SAS CODE

We all have our own way of doing things. For example, you may always want your program headers to have specific information, or you may comment blocks of code in your program a certain way. Maybe you need to run a snippet of code regularly, or if you can't always remember the correct options for a

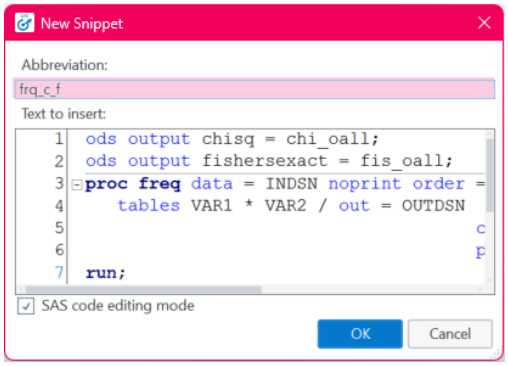
procedure, a quick way to insert these items is to create an abbreviation or a keyboard macro. These keyboard macros hold snippets of code that can be quickly called using a keyboard stroke. Creating a keyboard macro or abbreviation differs slightly between SAS Enterprise Guide and SAS Display Manager. We will illustrate both approaches.

To create an abbreviation and keyboard macro we can implement the following steps:

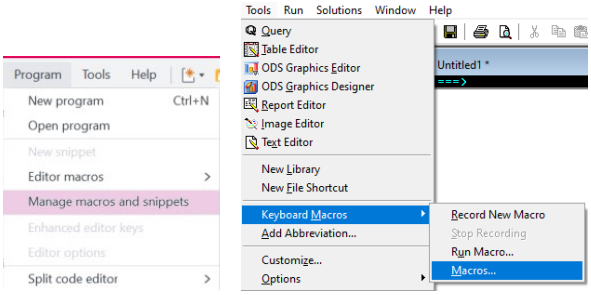
1. To create the abbreviation, we need to open the New snippet (SAS Enterprise Guide) or Add abbreviations (SAS Display Manager window.



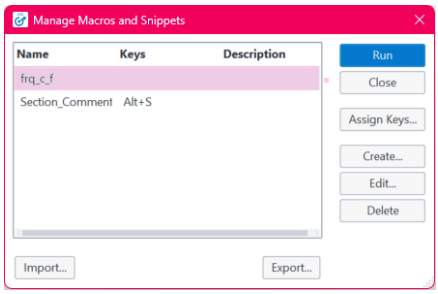
2. In the New snippet (Add Abbreviation) dialogue box, insert an abbreviation for the snippet of code or text string that is to be inserted into the program editor. In the text to insert box, the text string that is displayed when the abbreviation is typed is entered. For this example, we want to be able to bring up the syntax for the FREQ Procedure to produce a Chi-Square and Fisher's Exact test as well as produce a stacked plot.



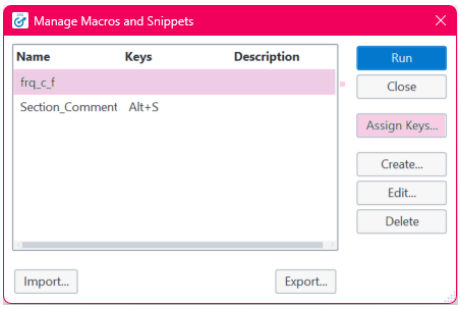
3. If you want to assign keystrokes to the abbreviation, you need to access the Macro Management window by selecting Program → Manage macros and snippets in SAS Enterprise Guide or Tools → Keyboard Macros → Macros in SAS Display Manager.



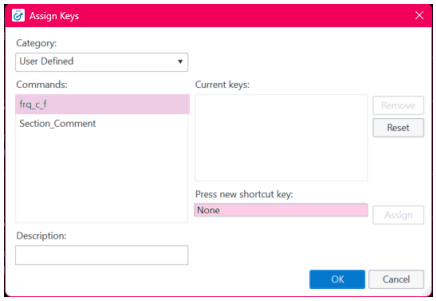
4. The abbreviation that was just created should appear in the list of user-defined macros and snippets.

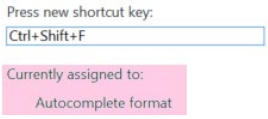


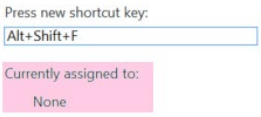
5. To assign the keystrokes, select Assign Keys.



6. Assign Keys dialogue box opens and you can select the keyboard macro that you want to assign keys (or change the assigned keys). Place the cursor in the box 'Press new shortcut key' and select the keys you want to use to invoke the snippet of code. Be mindful that some keystrokes already have pre-defined snippets. After you enter your keystrokes, it will indicate if the specific strokes are already assigned. If they have been, then you can choose to overwrite the pre-defined value (not recommended) or assign a different set of keystrokes. Select Assign and OK.

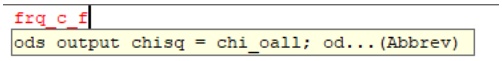






Now every time we enter 'frq_c_f' in the program editor or we use keystrokes, the insert command is invoked, and the string assigned is written to the program editor.

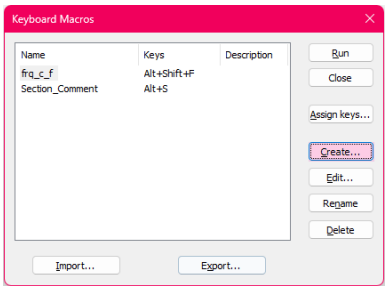
When you type in the abbreviation a pop-up is displayed showing a portion of the text that will be entered.



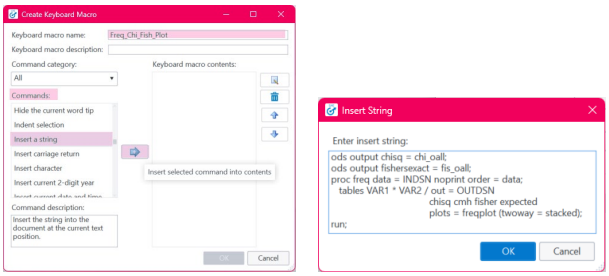
Upon hitting return, the full text is shown. The full text can also be retrieved by using the keystrokes that were assigned (Alt+Shift+F). In addition, you can open the macro manager and select the macro and then run to display the full text string.

```
ods output chisq = chi_oall;
ods output fishersexact = fis_oall;
proc freq data = INDSN noprint order = data;
   tables VAR1 * VAR2 / out = OUTDSN
                                chisq cmh fisher expected
                                plots = freqplot (twoway = stacked);
run;
```

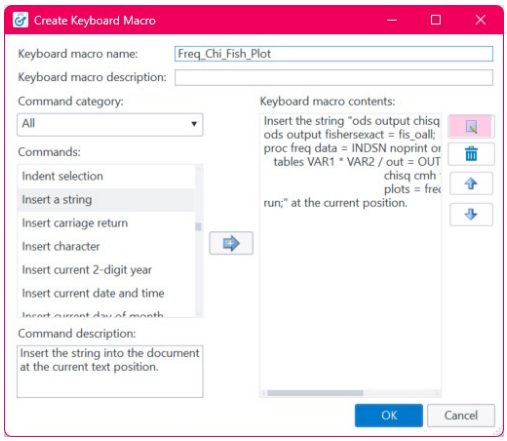
1. Note that if you do not wish to assign an abbreviation, you can skip steps 1 and 2 and go directly to the macro manager and create a new keyboard macro.



2. A new window appears, where you enter the name of the keyboard macro you are trying to create. You then need to select a command from the list of commands in the left pane. For this illustration, we are using 'Insert a string'. The arrow button in the middle is used to move the command to the right pane.



3. After you have entered, the string, select OK and this returns you to the 'Create Keyboard Macro' screen which now shows the command and the string. If you need to edit the string, you can select the notepad icon that is to the right of the keyboard macro content pane. This will re-open the 'Insert String' dialogue box. Once you are satisfied with the content, you select OK and this returns you to the macro management window.



4. After the keyboard macro is created, the keystrokes can be assigned by following steps 5 and 6.

CONCLUSION

This paper reviewed 10 tips that can enhance any SAS programmer's programming toolkit. These tips are just some of many ways to provide solutions to common programming problems. We recommend reviewing the tip and using it within your own data and SAS environment. These tips can provide a starting point to delve into various programming solutions. Many times, one doesn't know a solution exists until a problem occurs. The authors acknowledge that these tips are useful and are provided to assist other programmers, but they are ever evolving and can be enhanced. If you have questions or suggestions about any of these tips, feel free to contact the authors.

REFERENCES

Black, S. (2019). Keyboard Macros! An awesome tool you may have never heard of - once you do, you will never program the same again (It's that amazing!). Seattle: WUSS. Retrieved from https://www.lexjansen.com/wuss/2019/208_Final_Paper_PDF.pdf

Brill I., E. P. (2006). An Introduction to SAS Dictionary Tables - Paper 259-31. *Proceedings of the SAS Users Group International 31 Conference (SUGI 31).* San Francisco: SUGI.

CDISC Analysis Data Model Team. (2021, Nov). *Analysis Data Model Implementation Guide v1.3*. Retrieved from CDISC: https://www.cdisc.org/standards/foundational/adam

DelGobbo, V. (2006). Creating AND Importing Multi-Sheet Excel Workbooks the Easy Way with SAS - Paper 115-31. *Proceedings of the SAS Users Group International 31 Conference (SUGI 31)*. San Francisco: SUGI.

Hemedinger, C. (2015, May 20). *SAS Blogs*. Retrieved from SAS Dummy Blog: https://blogs.sas.com/content/sasdummy/2015/05/20/using-libname-xlsx-to-read-and-write-excel-files.

SAS Institute Inc. (2021, Oct 08). *Using Editor Macros*. Retrieved Jan 2023, from SAS® Enterprise Guide 8.1 documentation: https://documentation.sas.com/doc/en/egdoccdc/8.1/egug/n190dw6xkgmupxn1dfi0ldzt0tcj.htm

SAS Institute Inc. (2022, Sep 02). *ARRAY Statement*. Retrieved Jan 2023, from SAS® 9.4 and SAS® Viya® 3.5 Programming Documentation: https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/lestmtsref/p08do6szetrxe2n136ush727sbuo.htm

SAS Institute Inc. (2022, Aug 15). *Dictionary of Functions and CALL Routines*. Retrieved Jan 2023, from SAS® 9.4 and SAS® Viya® 3.2 Programming Documentation: https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.2/lefunctionsref/p1q8bq2v0o11n6n1gpij335fqpph.htm

SAS Institute Inc. (2022, Sep 30). *TRANSPOSE Procedure*. Retrieved Jan 2023, from SAS® 9.4 and SAS® Viya® 3.5 Programming Documentation: https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/proc/p1r2tjnp8ewe3sn1acnpnrs3xbad.htm

ACKNOWLEDGMENTS

The authors want to thank Lex Jansen for continuing to publish SAS Proceedings from 1976 to present. The website searches 36,515 papers (as of the date of writing this paper) and is a wealth of information for SAS Users. https://www.lexjansen.com

RECOMMENDED READING

- *Base SAS® Procedures Guide*
- *SAS® For Dummies®*

CONTACT INFORMATION

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

Lisa Mendez
Catalyst Clinical Research
lisa.mendez@catalystcr.com

Richann Jean Watson
DataRich Consulting
richann.watson@datarichconsulting.com