SAS® GLOBAL FORUM 2021

Paper 1041-2021

How SAS® 9.x enriched your SAS® programmer life?

Angelo Tinazzi, Cytel Inc.

ABSTRACT

SAS®9 is the longest-running version as it was first released in 2002, with its last technical maintenance release last summer (SAS 9.4 M7). In this presentation, I will go through some of the main Base SAS language enrichments you might have missed. For example, new functions to handle character variables, new DATA step options and new macro features. The focus will be on improvements that make work life better for me and every other SAS programmer.

INTRODUCTION

With this paper, I like to go through some of the major SAS/BASE programming language improvements introduced by SAS 9 some of you might have missed. The emphasis will be on changes or addition that made, in my opinion, my and every SAS programmer life, a better and happier programmer life.

A BIT OF SAS HISTORY

The first SAS version it was released almost 50 years ago (figure 1). Since then, what was called Statistical Analysis System, became an analytics platform supporting a wide range business.

The first versions of SAS were named after the year in which they were released. SAS 71 was published as a limited release; it was used only on IBM mainframes and had the main elements of SAS programming, such as the DATA step and the most common procedures in the PROC step. The following year a full version was released as SAS 72, which introduced the MERGE statement and added features for handling missing data or combining data sets.

Version 5 introduced a complete macro language, array subscripts, and a full-screen interactive user interface called Display Manager. In 1985 SAS was re-written in the C programming language; this allowed for the SAS' Multivendor Architecture that allows the software to run on UNIX, MS-DOS, and Windows. It was previously written in PL/I, Fortran, and assembly language[1].

Among the versions I had the pleasure to use, I like to mention the following:

- the first version on Windows platforms in early '90
- the availability of ODS in SAS 8 making a great enhancement for reporting
- the first SAS 9 released in 2002 making SAS 9 the longest-running version
- the latest version released in 2013, the SAS 9.4, with maintenance 7 being the latest technical maintenance released last August

[1] https://en.wikipedia.org/wiki/SAS_(software)

Moreover, starting with version 9.3 SAS started to also versioning the SAS/STAT module. For example, with the last SAS 9.4m7, 15.12 is the version of the SAS/STAT module.

Myself, I started a while go on VMS operating system where I did learnt SAS on version 5.

Highly likely, most if not all companies, are now running SAS 9.4 on different maintenances. If not, the graph on figure 2 from a very popular SAS blog could help you understanding how old and eventually how obsolete it is the SAS version you are currently running.

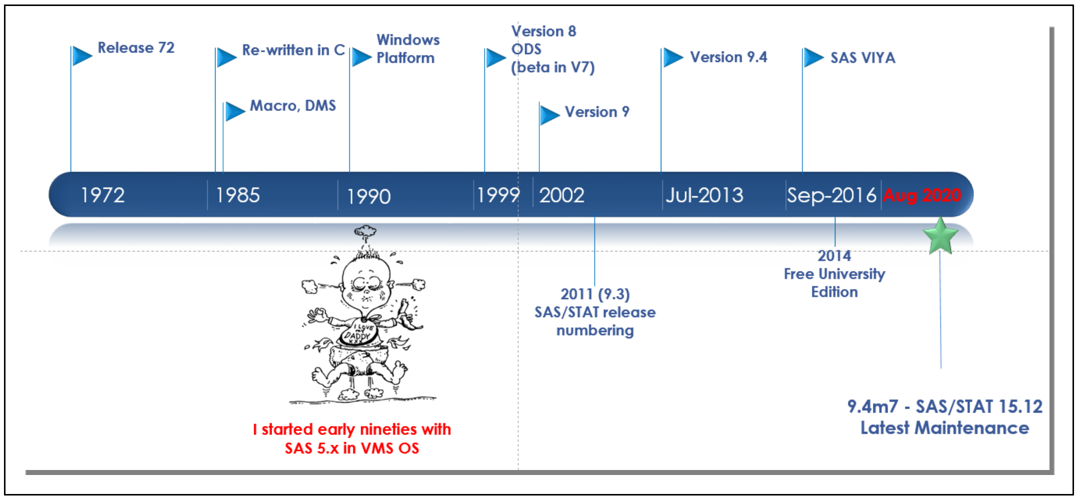

Release 72
Re-written in C
Macro, DMS
Windows Platform
Version 8 ODS (beta in V7)
Version 9
Version 9.4
SAS VIYA
1972
1985
1990
1999
2002
Jul-2013
Sep-2016
Aug 2020
2011 (9.3) SAS/STAT release numbering
2014 Free University Edition
I started early nineties with SAS 5.x in VMS OS
9.4m7 - SAS/STAT 15.12 Latest Maintenance

Figure 1: "Evolution" of SAS

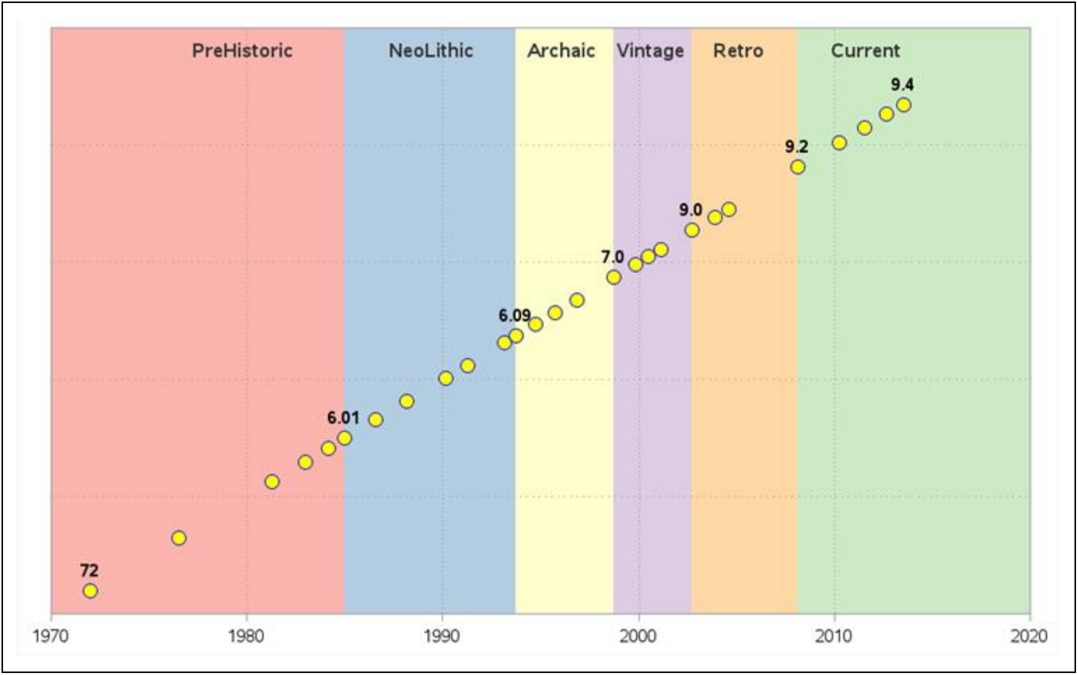
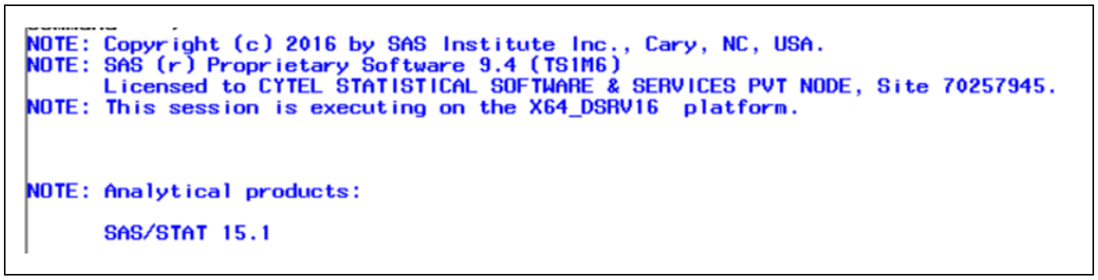


Figure 2: SAS Releases Timeline

https://blogs.sas.com/content/iml/2013/08/02/how-old-is-your-version-of-sas-release-dates-for-sas-software.html

At Cytel for example we are using the second-last released maintenance, the m6 (figure 3). If you want to check which version and maintenance you are currently running, you can check your SAS log when you start SAS or you can submit a PROC SETINIT.



```
NOTE: Copyright (c) 2016 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) Proprietary Software 9.4 (TS1M6)
      Licensed to CYTEL STATISTICAL SOFTWARE & SERVICES PVT NODE, Site 70257945.
NOTE: This session is executing on the X64_DSRV16  platform.


NOTE: Analytical products:

      SAS/STAT 15.1
```

Figure 3: Checking the SAS version you are running

SIGNIFICANT CHANGES IN SAS/BASE LANGUAGE

NEW FUNCTIONS

One of the thing I like more when a new SAS version is available is the availability of new functions.

These functions can improve the efficiency of your programs; things that may be before you were doing by developing either your own macros or by combining several functions and statements to get the desired outcome, became very often in the new release a function. The example I like more is the MEDIAN function that surprisingly was not available until SAS 9.

New character functions, search and compare character strings in addition to concatenating character strings, have been added in SAS 9; also, new functions that were available in other language/procedure like the COALESCE in PROC SQL are now available for use in the DATA step. Moreover, the entire family of the “V functions” or you have now a set of functions where you can better search for text patterns including counting occurrences, for example with the COUNT function.

The CAT Functions

Are programmers still using TRIM or LEFT or concatenation string? There is nothing wrong with it, but the code might be quite difficult to read when you need to concatenate several strings/variables. Now, with SAS 9 with the family of CAT functions, in particular the CATX function, not only you can shorten the code, as shown in the below example, you can also automatically eliminate redundant delimiters (Horstman, 2016). Moreover, numeric variables are by default converted to a string (BEST12 format) without a NOTE in the log.

```
Before SAS 9
old = put(n,1.)||” ”||trim(a)||” “||trim(b)||” “||c;

With SAS 9.x
new = CATX(“ “, n , a, b, c);
```

Table 1 summarizes the functioning of some of the main character stings.

Function	**Functioning**
TRIM	Remove leading blank

Function	Functioning
LEFT	Remove trailing blank
STRIP	Combine TRIM and LEFT
CAT	Concatenate string. Equivalent to !! or ¦¦
CATS	STRIPs both leading and trailing blanks, and does not insert separators
CATX	STRIPs both leading and trailing blanks, and inserts separators. The first argument to CATX specifies the separator

Table 1. Key Character functions

ANY and NOT Functions

The family of ANY functions (Horstman, 2017), and their opposite NOT functions, could facilitate your code in detecting if a string contains a 'family' of characters i.e. any alphabetic characters as in the in the case of the function ANYALPHA or use the ANYUPPER function to check if a string contains a capitalized letter. See also the following example:

```
With SAS 9
if LBORRES ne “” and not(ANYALPHA(LBORRES)) and
                 not not(ANYSPACE(strip(LBORRES))) and
                 indexc(LBORRES, “<>+-“) eq 0 then …..

Before SAS 9
if LBORRES ne “” and indexc(upcase(LBORRES),
                            “QWERTZUIOPASDFGHJKLYXCVBNM...”))
   eq 0 then ....
```

The IFC function, and its numeric version, provides a convenient way to package conditional logic right into a function call (Kesireddi, 2017). The function accept a logical expression as its first parameter:

- If the logical expression evaluates to true, it returns the value of the second argument
- If the logical expression evaluates to false, it returns the value of the third argument
- If the logical expression is missing, it can return the value of the optional fourth argument.

For example, assumes you want to create a Boolean variable to identify 'white' subjects to use in your regression statistical model, you can use the IFN function as follows:

```
ARACE=IFN(RACE=”WHITE”,1,0);
```

Two CHOOSEC function (and CHOOSEN) can also shorten lengthy line of codes. From the example provided by (Kesireddi, 2017), supposed you want to assign to the ADaM variable APERIODC a value based on the value of the variable APERIOD. With the traditional IF/THEN/ELSE IF statements, you would have coded as follows:

```
if APERIOD=1 then APERIODC='INDUCTION';
else if APERIOD=2 then APERIODC='MAINTENANCE';
else if APERIOD=3 then APERIODC='TREATMENT';
else APERIODC='';
```

With the CHOOSEC function the above statements can be replaces by a single line of code as follows:

```
APERIODC=CHOOSEC(APERIOD, 'INDUCTION', 'MAINTENANCE', 'TREATMENT', '');
```

The CHOOSEC function returns a character value that represents the results of choosing from a list of arguments, so to each of the value of the first integer argument, APERIOD, is assigned the equivalent value from the list of specified selected arguments: 1='INDUCTION', 2='MAINTENANCE' and 3='TREATMENT'.

The WHICHC function and its numeric equivalent WHICHN function, returns instead the position within a list of arguments from the result of an expression. For example, let revert the previous example and suppose we want, based on the value of APERIODC, assign an integer value to APERIOD

```
APERIOD=WHICHN(APERIODC, 'INDUCTION', 'MAINTENANCE', 'TREATMENT', '');
```

APERIOD will be equivalent to 1 if APERIODC is ‘INDUCTION’, 2 if equivalent to ‘MAINTENANCE’ and 3 if equivalent to ‘TREATMENT’, Null otherwise.

INDEX vs FIND vs COUNT

The INDEX function was historically used to search for a string occurrence in a string. With SAS 9 two functions have been added to either find specific position of occurrence, the FIND function, or to count the number of occurrences of a string in a character variable, the COUNT function. See here an example.

`data _null_;`	
`  str=”abc abc abc”;`	
`  ind=index(str,”abc”);`	`1 (position 1st occurrence)`
`  fnd1=find(str,”abc”);`	`1 (position 1st occurrence)`
`  fnd2=find(str,”abc”,4);`	`5 (position 2nd occurrence)`
`  cnt=count(str,”abc”);`	`3 (nr of occurrences)`
`run;`	

COALESCE and COALESCEC

The COALESCE function (and COALESCEC for character variables) works same as the COALESCE function available in PROC SQL as it takes the value of the first non-missing 'argument' in the list of variables specified (Horstman, 2016).

```
With SAS 9
htBL=COALESCE(of htbl3-htbl1);

Before SAS 9.x
if htbl3 ne . Then htBL=htbl3;
Else if htbl2 ne . Then htBL=htbl2;
Else if htbl1 ne . Then htBL=htbl1;
```

The V Functions

The family of V functions allows programmers to query variable attributes within the data step without having to "hard code" the information in the code or without having to do a preliminary step to incorporate data from PROC CONTENTS for example (Watson and Miller, 2015).

In the following example, assume you have some variables with an assigned user format to denote the meaning of the numeric code, you can automatically create, also using an array, a set of variables containing the decoded version by dynamically query the format attribute using the VFORMAT function.

```
data demo_decode;
  set demo;
  array ORIGvar   (2)   racen sexn;
  array DECODEvar (2) $ race  sex;
  do i=1 to 2;
     DECODEvar(i)=putn(ORIGvar(i), VFORMAT(ORIGvar(i)));
  end;
run;
```

Of note the DECODEvar(s) assignment can be also shorten with the use of the VVALUE function

```
DECODEvar(i)=VVALUE(ORIGvar(i));
```

New Numeric / Stats Functions

There are also several new descriptive statistics and mathematical functions. An example are the MEDIAN and the GEOMEAN functions, to calculate the median or geometric mean respectively, which were not available prior to version 9; or the LARGEST function where you can specify for example you want to get the second largest values of among several values.

```
Before SAS 9
Array? Transpose? SET BY?

With SAS 9.x
LARGEST(2,of sf1-sf4,sf8);
```

Functions Modifiers

Another nice feature with SAS functions, which I think added in SAS 9.1, is the ability of using the “function modifiers”, as an additional argument to modify the normal behavior of a SAS function. It is the case for example of the COMPRESS function, a function historically used to remove all spaces from a string. The function now, not only accept a second argument where you can specify the list of characters you want to remove or compress, but you can through the third argument, the modifier, specify a complete set of ‘family’ of characters to remove. In the following example, the ‘p’ modifier in the compress function will remove all punctuations from the specified string.

```
STRINGNEW=compress(STRING,,”p”);
```

OTHER “COOL” IMPROVEMENTS AND OPTIONS

Among new syntax features added in SAS 9, I like the use of some sort of shortcuts.

One example is the use of a colons (:) in the SET command in SAS 9.2, where you can use the colons as a wildcard to set all datasets starting with the same prefix, like shown in the following example where all SDTM supplemental qualifier datasets are set together

```
set supp:;
```

Or a range of datasets using the “-“

```
set lb1 - lb50
```

The IN Boolean operator now allows some shortcut such as the specification of a range of values. For example

```
IN(2 3 5:10)
```

if used it will be true when the variable or expression will be equal to 2,3,5,6,7,8,9,10.

SAS 9.2 also introduced new formats and informats, such as E8601DT and E8601DA, to support ISO 8061 standard format for date, time and intervals required in CDISC SDTM (Morgan, 2017).

The HASH Objects

The addition of the HASH objects in SAS 9.1 represents an excellent feature for lookup facilities, especially when you have to deal with big datasets; this is because the HASH objects are the first in-memory data structure accessible from the DATA step (Allen, 2019)(Dorfman and Enderson, 2018).

For example, I did a test to compare the use of HASH objects against two traditional alternatives in SAS to search and extract from a large dataset e.g. 20 millions of records. From table 2, you can see from the simple test how fast are the HASH objects compared for example to an SQL or the traditional MERGE statement.

SQL [~23 seconds]	**MERGE** [~36 seconds]	**HASH** [~14 seconds]

proc sql noprint;
  create table SQL_ as
    select *
    from lb
    where lbtestcd in
     (select distinct
      lbtestcd from LB
      where lbtoxgr>0);
quit;
	proc sql noprint;
	  create table grades as
	    select distinct lbtestcd
	    from LB(where=(lbtoxgr>0));
	quit;
	proc sort data=LB out=LB_sort;
	  by lbtestcd;
	run;
	data MERGE_;
	  merge LB_sort
	        grades (in=ingr);
	  by lbtestcd;
	  if ingr;
	run;
		data HASH_;
		  if _n_=1 then do;
		    dcl hash g(dataset:"LB(
		        where=(lbtoxgr>0))");
		    g.definekey("lbtestcd");
		    g.definedone();
		  end;
		  set LB;
		  if **g.find()=0;**
		run;

Table 2. HASH vs SQL vs MERGE

However, in my opinion this technique should be used at your own risk especially if the code must be understood by a reviewer in the event your code is part of a submission package, given the fact the use of HASH objects requires the use of some, let me say, non-traditional SAS syntax as highlighted in the above example.

The PERL Regular Expressions

The Perl Regular expressions added in SAS 9 (Ni and Burmenko, 2019), although similar SAS regular expressions were available in earlier versions (Cody and Wood, SUGI 29), are an enormously powerful technique for advanced “string pattern search”. Many of this string processing tasks can be achieved using the traditional character functions, but the Perl regular expressions provide a more compact and efficient solution.

For example, you could write a regular expression to look for four digits, a dash, two digits, again a dash, followed by two digits, to check if a date variable in an SDTM dataset is a complete date and in ISO format before converting it into a SAS date. See here the equivalent code

```
if PRXMATCH("/\d{4}-\d{2}-\d{2}/o", aestdtc)=1 then
   asdt = input(substr(aestdtc,1,10),yymmdd10.);
```

Again, like for the “HASH objects”, use this technique at your own risk.

SIGNIFICANT CHANGES IN SAS MACRO LANGUAGE

Let have now have a look at some new cool stuff available now in the macro language.

Use of %IF%THEN in open code

The first one worth to mention is a relatively new one as it became available with one of the latest maintenance, the SAS 9.4 maintenance 5 and, as I previously mentioned, depending on your company policy regarding software update, it could be that you have earlier version installed at your site and therefore this functionality will not work. This is about the ability of running the %IF/%THEN macro statements in open code, so outside the %MACRO%MEND wrapper. This could be particularly useful in situations like where you want your SAS code conditionally executed depending on the occurrence of a specific event.

See the following example where, before making the PROC MEANS, I wanted to make sure the dataset ADLB exists, for example if a previous part of code is ended with an error and therefore the ADLB not generated.

```
%if %sysfunc(exist(work.adlb)) %then %do;
    proc means data=work.adlb;
    run;
%end;
%else %do;
    %PUT WARNING: Missing WORK.ADLB - Please check;
%end;
```

There are however few limitations. For example, your %IF/%THEN must be followed by a %DO/%END block for the statements that you want to conditionally execute and the same is true for any statement that follow the optional %ELSE branch of the condition.

Furthermore, you cannot have any nesting of multiple %IF/%THEN constructs. If you need that flexibility, you can do that within a %MACRO wrapper instead.

New IN Boolean Operator

Another interesting improvement in SAS macro in SAS 9, it is the availability of the IN boolean operator.

For example, suppose you want to check in your macro if the user has correctly specified a required parameter, meaning that the selected parameter has a valid and expected value. Prior to SAS 9 you would need to check each individual value with an OR operator, while now with SAS 9 you can make use of the IN operator and therefore shorten your code.

```
Before SAS 9
%macro mymacro(option=);
  %if &option eq DEBUG or &option eq STORE or
      &option eq REPORT %then ...


With SAS 9
macro mymacro(option=)/ minoperator mindelimiter=”,”;
  %if &option IN(DEBUG,STORE,REPORT) %then ...
```

New Macro Functions

The following three macros functions became available in SAS 9:

- The **%SYMEXIST** function returns an indication of the existence of a macro variable

- The **%SYMGLOBL** function returns an indication as to whether a macro variable is global in scope
- The **%SYMLOCAL** function returns an indication as to whether a macro variable is local in scope

NEW ALTERNATIVES TO MACRO AND PROGAMMING TECHNIQUES

There are now several alternatives to macro in SAS 9 that let you chose among more advanced technologies for problem solving and data manipulation. Table two lists some of these recent technologies (see also references at the end of the paper).

Technique	**Description**
PROC FCMP PROC PROTO [SAS 9.2]	To create user functions/call routines (Boyko, 2020)
PROC GROOVY [SAS 9.3]	Java syntax compatible object-oriented programming (Kennedy, 2019)
PROCS DS2 [SAS 9.4]	SAS proprietary object-oriented programming language For advanced problem solving and data manipulation (Voieyutkyi, 2017)
PROC LUA [SAS 9.4]	Lua Language. This could also expand SAS functionalities for application development to overcome SAS macro limitations for example (Tomas, 2015)

Table 3. New "Programming Techniques" available with SAS 9.x

DOSUBL

This is not really an alternative to SAS macro language, but introduced with SAS version 9.3M2, with the DOSUBL function you have now the "power" to submit code to SAS and wait for that code to complete submission. That way you can now execute a DATA step inside of another DATA step or write "function-style" macros executing complete SAS language statements; that was not possible with traditional SAS macro programming. This is now possible because the DOSUBL submit SAS code to run "on the side" while your SAS DATA step is still running as well explained by McMullen at the latest SAS Global Forum (McMulle, 2020). This "behavior" distinguish the DOSUBL function from another yet powerful technique that allow the writing of dynamic code within a DATA step, the CALL EXECUTE.

One of the limitation of the CALL EXEUCUTE was that the code can be not executed until after that DATA step that invoked CALL EXECUTE has completed. The classic example to compare DOSUBL function vs CALL EXECUTE is reported in the following examples taken from the paper of Langston (Langston, 2013).

In the below example, the PUT statement in the DATA step does not resolve to ‘a’ as someone might think, giving the fact the value ‘a’ is passed as a parameter to the %doit macro. However, since the macro %doit is executed after the end of the DATA step, the macro variable ‘xyz’ used in the PUT statement is resolved to ‘q’, that was the value assigned to the macro variable ‘xyz’ with the %LET statement prior to the DATA step containing the CALL EXECUTE statement.

```
%macro doit(value);
  data _null_;
    call symput('xyz',"&value.");
  run;
%mend doit;
%let xyz=q;
data _null_;
  call execute('%doit(a)');
  xyz_value = symget('xyz');
  put xyz_value=;
run;
```

On the contrary, with the use of the DOSUBL function, the macro variable ‘xyz’ used in the SYMGET function in the DATA step, get resolved to the value resolved by the CALL SYMPUT executed inside the %doit macro and executed in the DATA step prior to the SYMGET function.

```
%macro doit(value);
  data _null_;
    call symput('xyz',"&value.");
  run;
%mend doit;
%let xyz=q;
data _null_;
  rc = dosubl('%doit(b)');
  xyz_value = symget('xyz');
  put xyz_value=;
run;
```

To give you another example to better understand the power of DOSUBL, let us consider a situation where we want to create a dataset containing some summary statistics, the rate, on the occurrence of a specific event under different conditions, for example the clinical trial arm and the study population assigned.

I would need two macros, one for extracting the denominator and one for extracting the numerator e.g. the occurrence of a specific condition (see figure 4 and 5)

```
/*Macro to Get Denominator*/
%macro getDen(in=,where=);
  %local rc den;
  %let den=0;
  %let rc=%sysfunc(dosubl(%nrstr(
    proc sql noprint;
      select count(*) into :den from &in where &where;
    quit;
  )));
  &den
%mend getden;
```

Figure 4: Macro to get Denominator

```
/*Macro to Get Numerator*/
%macro getCount(in=,where=);
  %local rc count;
  %let count=0;
  %let rc=%sysfunc(dosubl(%nrstr(
    proc sql noprint;
      select count(*) into :count from &in where &where;
    quit;
  )));
  &count
%mend getCount;
```

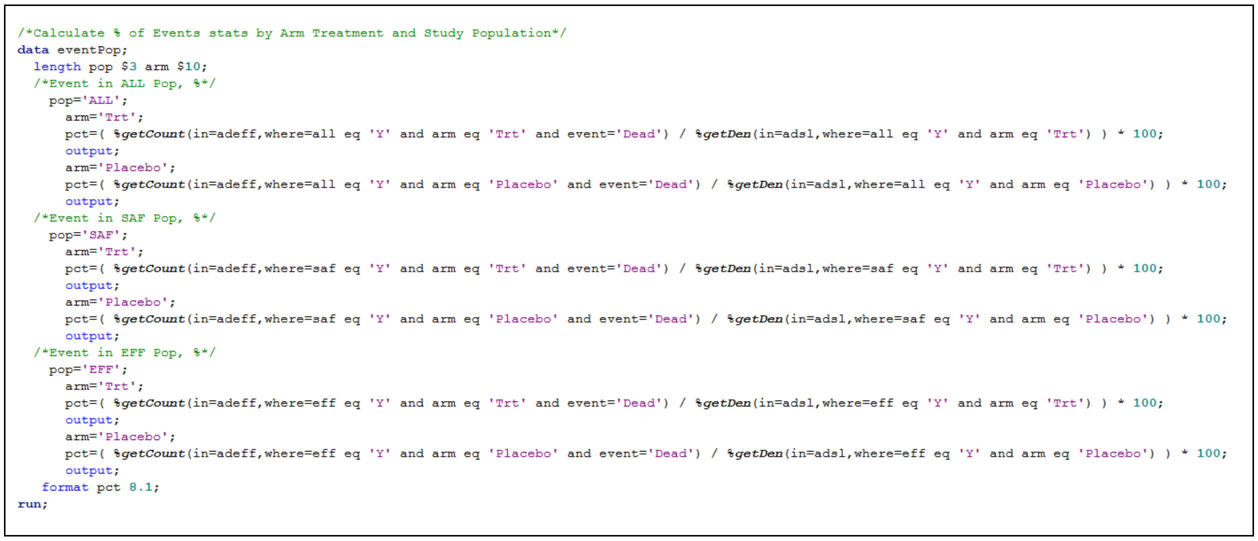
Figure 5: Macro to get Numerator

The DOSUBL syntax in this instance when used in a SAS macro, it simply is the DOSUBL function followed by a combination of SAS code, either macro function or data and proc steps. The core of the two macros is the PROC SQL and the return of a specific macro parameter, respectively the denominator, *&den*, and the numerator, *&count*.

I would then need to call the two macros several times to build my *eventPop* dataset, containing one record per each level of summarization, where *pct* variable will contain the proportion or % of subjects having the death event on each specific subset

The variable containing the % will need first to get the number of subjects with that condition, for example number of subjects dying in treatment arm in ALL population and the denominator on which the % will be calculated.

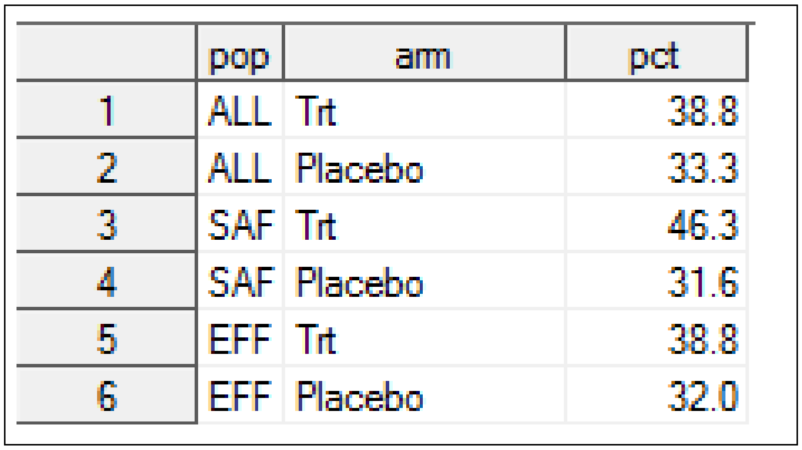
You can note that the call of the two macro is equivalent to call two separate PROC SQL per pct calculation, and this is while the data step is still running. The result of the two proc sql from each of the two macros call is then given back to the data step to complete the calculation of the pct variable. See the code in figure 6.



```
/*Calculate % of Events stats by Arm Treatment and Study Population*/
data eventPop;
  length pop $3 arm $10;
  /*Event in ALL Pop, %*/
    pop='ALL';
      arm='Trt';
      pct=( %getCount(in=adeff,where=all eq 'Y' and arm eq 'Trt' and event='Dead') / %getDen(in=adsl,where=all eq 'Y' and arm eq 'Trt') ) * 100;
      output;
      arm='Placebo';
      pct=( %getCount(in=adeff,where=all eq 'Y' and arm eq 'Placebo' and event='Dead') / %getDen(in=adsl,where=all eq 'Y' and arm eq 'Placebo') ) * 100;
      output;
  /*Event in SAF Pop, %*/
    pop='SAF';
      arm='Trt';
      pct=( %getCount(in=adeff,where=saf eq 'Y' and arm eq 'Trt' and event='Dead') / %getDen(in=adsl,where=saf eq 'Y' and arm eq 'Trt') ) * 100;
      output;
      arm='Placebo';
      pct=( %getCount(in=adeff,where=saf eq 'Y' and arm eq 'Placebo' and event='Dead') / %getDen(in=adsl,where=saf eq 'Y' and arm eq 'Placebo') ) * 100;
      output;
  /*Event in EFF Pop, %*/
    pop='EFF';
      arm='Trt';
      pct=( %getCount(in=adeff,where=eff eq 'Y' and arm eq 'Trt' and event='Dead') / %getDen(in=adsl,where=eff eq 'Y' and arm eq 'Trt') ) * 100;
      output;
      arm='Placebo';
      pct=( %getCount(in=adeff,where=eff eq 'Y' and arm eq 'Placebo' and event='Dead') / %getDen(in=adsl,where=eff eq 'Y' and arm eq 'Placebo') ) * 100;
      output;
   format pct 8.1;
run;
```

Figure 6: Calling "function-style" macros in a datastep

The resulting dataset will have one record per each study arm per each study population (figure 7).



	pop	arm	pct
1	ALL	Trt	38.8
2	ALL	Placebo	33.3
3	SAF	Trt	46.3
4	SAF	Placebo	31.6
5	EFF	Trt	38.8
6	EFF	Placebo	32.0

Figure 7: EventPop Dataset

CONCLUSIONS

There are many others enhancements available in SAS/BASE I did not cover in this paper. I have for example in mind all the new ODS destinations and functionalities added in SAS 9 together with ODS graphics or the improved integration with Ms Office applications and of course the many new or improved procedures. I have also not covered the SAS programming cloud revolution introduced by SAS VIYA (SAS Institute, 2019) (Shankar, 2020), but unfortunately I am not yet a SAS VIYA user.

I would also like to reference a nice resource to, not only challenge your skills, but also update your SAS skills: the Sasensei website (https://sasensei.com/). This is a free question-based learning system, and it was extremely helpful for refreshing my SAS skillsets and major source of inspiration for this paper.

REFERENCES

J. Horstman. 2016. *Let the CAT Out of the Bag: String Concatenation in SAS® 9*. SCSUG

J. Horstman. 2017. *Fifteen Functions to Supercharge Your SAS® Code*. WUSS

G. Kesireddi. 2017. *Efficient Coding Techniques In SAS*. PHUSE

R. Watson and K. Miller. 2015. *'V' for … Variable Information Functions to the Rescue*. MWSUG

D. Morgan. 2017. *ISO 8601 and SAS®: A Practical Approach*. PharmaSUG

R. Allen. 2019. *Innovative Clinical Programming Methods*. PharmaSUG-China

P. Dorfman and Don Enderson. 2018. *Data Management Solutions Using SAS HASH Table Operations*

Q. Ni, P. Burmenko. 2019. *Four steps to get a quick start with Perl Regular Expressions in SAS®*. PharmaSUG-China

R. Cody and R. Wood. *An Introduction to Perl Regular Expressions in SAS 9*. SUGI 29

I. Boyko. 2020. *PROC FCMP or Function you Create, Master and Proceed*. PHUSE

K. Kennedy. 2019. *A GROOVY way to enhance your SAS life*. PHUSE

S. Voievutkyi. 2017. *Expansion of Opportunities in Programming: DS2 Features and Examples of Usage Object Oriented Programming in SAS*. PharmaSUG

P. Tomas. 2015. *Driving SAS® with Lua*. SAS GLOBAL FORUM

Q. McMullen. 2020. *A Close Look at How DOSUBL Handles Macro Variable Scope*. SAS Global Forum

R. Langston. 2013. *Submitting SAS® Code On The Side*. SUGI

SAS Institute. 2019. *Exploring SAS® Viya® Programming and Data Management*. https://support.sas.com/content/dam/SAS/support/en/books/free-books/exploring-sas-viya.pdf

C. Shankar. 2020. *Coding in SAS® Viya®*. 2020

RECOMMENDED READING

- *SAS Functions by Examples – R. Cody – Second Edition 2010 [Book]*
- *Unstructured Data Analysis – K.M Windham – 2018 [Book]*
- *Modernizing Legacy SAS® Applications and Program Code KP. Lafler and C. Roberts; SCSUG 2017*
- *Comparing 6 Techniques To Do Data Driven Programming – J. Derks – PhUSE 2017*

CONTACT INFORMATION

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

Angelo Tinazzi
Cytel Inc.
Route de Prè-Bois 20
1215 Geneva – Switzerland
angelo.tinazzi@cytel.com
www.cytel.com