AN INTRODUCTION TO THE SAS MACRO LANGUAGE

Jonas V. Bilenas

March 12th, 2018

PhilaSUG

WHAT ARE SAS MACROS:

WHAT ARE SAS MACROS:
3 SAS MACRO GURUS

CONCEPT of SAS MACROS
Art Carpenter

- https://support.sas.com/en/books/authors/art-carpenter.html
- https://www.sas.com/store/books/categories/usage-and-reference/carpenter-s-complete-guide-to-the-sas-macro-language-third-edition/prodBK_67815_en.html
- Check out YOUTUBE Art Carpenter SAS GURU

CONCEPT of SAS MACROS

- From Art Carpenter's Complete Guide to the SAS Macro Language, 2nd edition:
  - "Macro language provides tools that allow you to:
    - Pass information between SAS steps.
    - Dynamically create code after the user submits the program for execution.
    - Conditionally execute DATA or PROC steps.
    - Create generalizable and flexible code."

CONCEPT of SAS MACROS
Ron Fehd

- Macro Design Ideas: Theory, Template, Practice Ronald J. Fehd, Stakana Analytics, Atlanta, GA, USA
  - https://support.sas.com/resources/papers/proceedings14/1899-2014.pdf

Introduction

Overview

This section review the context in which macros are written.

Who? : Who writes macros? Either users or programmers write macros.

What? : What is a macro? A macro is a program which can replicate statements around values supplied in parameters.

Why? : Why write macros? These are some reasons to create macros.

1. reuse
2. encapsulate complexity
3. to use either of the macro statements `%do` or `%if`
4. to use either of the macro functions `%sysevalf` or `%sysfunc`

When? : When are macros written, or polished? Macros are written after ad hoc programming produces several examples of similar processing that can be simplified into a macro. Polishing is best accomplished before peer review.

Where? : Where are macros? Macros occur in these places:

- within a program
- in a program in a project
- in a site folder available to all projects

Summary

These ideas on macro design are for programmers writing or polishing macros for use in either a project or site.

2nd page form:
Macro Design Ideas: Theory, Template, Practice Ronald J. Fehd, Stakana Analytics, Atlanta, GA, USA

- https://support.sas.com/resources/papers/proceedings14/1899-2014.pdf

CONCEPT of SAS MACROS
Ian Whitlock

- https://www.lexjansen.com/nesug/nesug04/hw/hw10.pdf Whitlock, I. & McMullen, Q. Macro Power, (2004)

CONCEPT of SAS MACROS

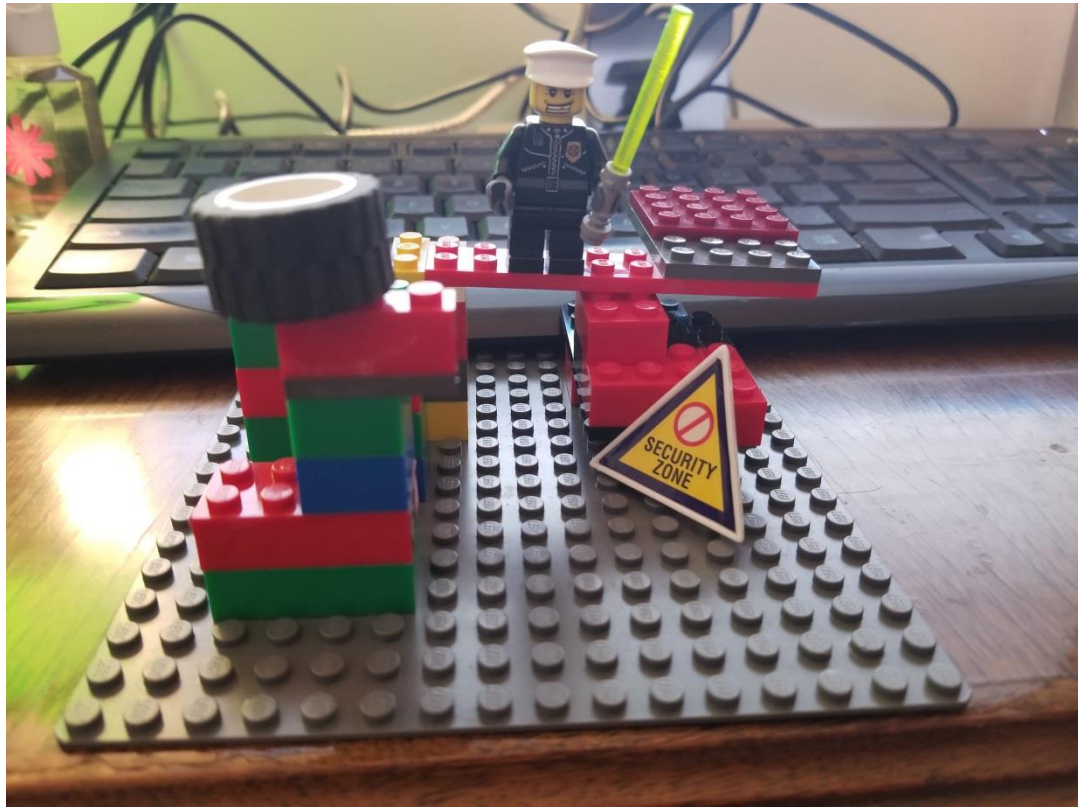
- From Ian Whitlock RUG papers:

  - “The beginner often mistakenly gets the impression that SAS macro is sort of a super SAS programming language. It isn't. It is a language for manipulating text to make SAS programs.”

  - “SAS has developed in a context where variable names are known and tied to specific data elements. This makes it easy to quickly develop small programs that can accomplish a lot. However, the programs are typically applicable to a very specific situation. When given a new dataset, the LIBNAME must change and the member name must change and all the usage of variable names may no longer be appropriate. In a word, SAS programs can be simple, but brittle. Many changes may be needed when the situation changes only slightly.

  - SAS programs may make good templates for particular problems, but they are not general solutions. They usually require some macro elements to make them general solutions to class of related problems. The main significance of the macro language is that it can return some of the flexibility to the SAS language needed to make general programs. A macro may be thought of as a parameterized unit of SAS code.”

My CONCEPT of SAS MACROS



CONCEPT of SAS MACROS

- You realize that the SAS code you are composing has certain steps or procedures that are similar and are repeated a number of times in your code. It would be nice if you don't have to copy and paste the code over and over and just call in a MACRO that you created to run the same code with provided user parameters that change how the code is processed.

- The MACRO code can take certain parameters (MACRO VARIABLES) which will modify the code when it executes.

    - Examples of MACRO VARIABLES:
        - The **data set** you want the MACRO to process.
        - Possible **WHERE** conditions.
        - **OPTIONS** used in PROCS.

- MACRO VARIABLES and MACRO CODE can be useful in Production Code.
    - Users don't change the code per say but change the parameters that generate new code and new output.
    - Useful when code is approved by CONTROLS or REGULATORY groups.

CONCEPT of SAS MACROS

- We will review:
  - MACRO VARIABLES
    - %LET
    - SYMPUT or SYMPUTX, that is the question.
    - INTO in squeal code (SQL).
  - MACRO CODE:
    - %IF, %THEN. %ELSE
    - LOOPING WITH %DO, %END
    - SOME MACRO FUNCTIONS:
      - %STR
      - %SYSFUNC
      - %UPCASE
      - %SCAN after squeal
  - WHERE YOU CAN STORE MACROS
    - AUTOCALL Directories
    - COMPILED MACRO CATALOGS
  - A few tricks to decode the LOG if there are ERRORS

A SIMPLE MACRO: Example 1

IMPORTING AN XLS FILE INTO A SAS DATA SET

Code to import an XLS file into SAS Data Set

```
PROC IMPORT OUT = NPV
            DATAFILE = 'baseline_input.xls'
            DBMS = XLS REPLACE;
     GETNAMES=YES;
     DATAROW=2;
     SHEET = "INPUTS";
     MIXED=NO;
RUN;
```

IMPORTING AN XLS FILE INTO A SAS DATA SET

- But you need to run the code a number of times.
- You have many tabs you want to read into SAS data sets
- Can you MACROSIZE the code? Will be nice to have around. Use in current production code you are working on and other times you code calls to import from an XLS file.

IMPORTING AN XLS FILE INTO A SAS DATA SET

THE MACRO CODE:

```
%macro import(out,datafile,DBMS,sheet);
PROC IMPORT OUT= &out.
            DATAFILE = "&datafile."
            DBMS = &DBMS. REPLACE;
     GETNAMES=YES;
     DATAROW=2;
     %if &DBMS. ne CSV %then %do;
       SHEET="&sheet.";
     %end;
     MIXED=NO;
RUN;
%mend;
```

OUT, DATAFILE, DBMS, SHEET are LOCAL MACRO VARIABLES.
Only available during the MACRO run.

IMPORTING AN XLS FILE INTO A SAS DATA SET

EXECUTING THE MACRO CODE:

```
%import(out=NPV,
        datafile = baseline_input.xls,
        DBMS=XLS,
        sheet=INPUTS);
```

IMPORTING AN XLS FILE INTO A SAS DATA SET

OR,
EXECUTING THE MACRO CODE:

```
%import(NPV,baseline.xls,XLS,INPUTS);
```

IMPORTING AN XLS FILE INTO A SAS DATA SET

Anything wrong with this code?

```
%import(NPV,baseline.xls,XLS,INPUTS)
```

IMPORTING AN XLS FILE INTO A SAS DATA SET

Wait. What if user enters lower case values for macro variable DBMS?

```
%macro import(out,datafile,DBMS,sheet);
PROC IMPORT OUT= &out.
            DATAFILE = "&datafile."
            DBMS = &DBMS. REPLACE;
     GETNAMES=YES;
     DATAROW=2;
     %if &DBMS. ne CSV %then %do;
       SHEET="&sheet.";
     %end;
     MIXED=NO;
RUN;
%mend;
```

IMPORTING AN XLS FILE INTO A SAS DATA SET

Wait. What if user enters lower case values for macro variable DBMS?

```
%macro import(out,datafile,DBMS,sheet);
PROC IMPORT OUT= &out.
            DATAFILE = "&datafile."
            DBMS = &DBMS. REPLACE;
     GETNAMES=YES;
     DATAROW=2;
     %if %UPCASE(&DBMS.) ne CSV %then %do;
       SHEET="&sheet.";
     %end;
     MIXED=NO;
RUN;
%mend;
```

IMPORTING AN XLS FILE INTO A SAS DATA SET

Some more user error checks:

```
%macro import(out,datafile,DBMS,sheet);
PROC IMPORT OUT= &out.
            DATAFILE = "&datafile."
            DBMS = &DBMS. REPLACE;
     GETNAMES=YES;
     DATAROW=2;
     %if %UPCASE(&DBMS.) ne CSV %then %do;
       SHEET="&sheet.";
     %end;
      %else if %UPCASE(&DBMS.) ne CVS %then %do;
       SHEET="&sheet.";
     %end;
     MIXED=NO;
RUN;
%mend;
```

MACRO VARIABLES

MACRO VARIABLES

- MACRO VARIABLES
  - MACRO VARIABLES in SAS can be defined in a number of ways.
    - Using the **%let** statement in open code.
    - Using SQL **INTO** statement.
    - Using **CALL SYMPUT or CALL SYMPUTX** in a DATA STEP, typically in a DATA _NULL_ step.
    - Use **ODS OUTPUT** along with CALL SYMPUT to capture elements of SAS PROCEDURE Output

  MACRO variables can be used downstream after assignment in subsequent open code if defined as GLOBAL.

Naming MACRO VARIABLES

- Name can be from 1 to 32 characters in length.
- Must begin with a letter or underscore (_).
- Any Combination of letters and numbers may follow.
- Text that is stored in a macro variable can be up to 64K in length.
- Reference macro variables with an ampersand (&) prefix.
- You can put a period after the macro names. This will avoid confusion when concatenating text after the macro variable.
- Easiest way to define a macro variable is to use the %let statement:

```
%let stuff=bourbon;
```

- Display macro variables:

```
%put the macro variable stuff is &stuff.;
```

EXAMPLE 2: Using MACRO Variables in Open Code.

MACROS in SAS. MACRO VARIABLE ASSIGNMENT

- **MACRO VARIABLE ASSIGNMENT EXAMPLES: %LET**

```
options nocenter mprint symbolgen fullstimer;

%let ds=adv_sas;
%let var=years;

data &ds.;
  input &var @@;
  label years = 'Years of SAS Experience';
  datalines;
  3 1.5 .5 6 5 7 7 3.5 3.5 10 5.5 16 7 .5 5
  2 1 6.5 3.5 25 10 3 1 3.5
  ;;
run;
```

The 2 macro variables are global macro variables. They can be used anywhere in the code; open code or macro code.

MACROS in SAS. MACRO VARIABLE ASSIGNMENT

WAIT. WAIT, WAIT, WHAT ABOUT ANOTHER MACOR VARIABLE?

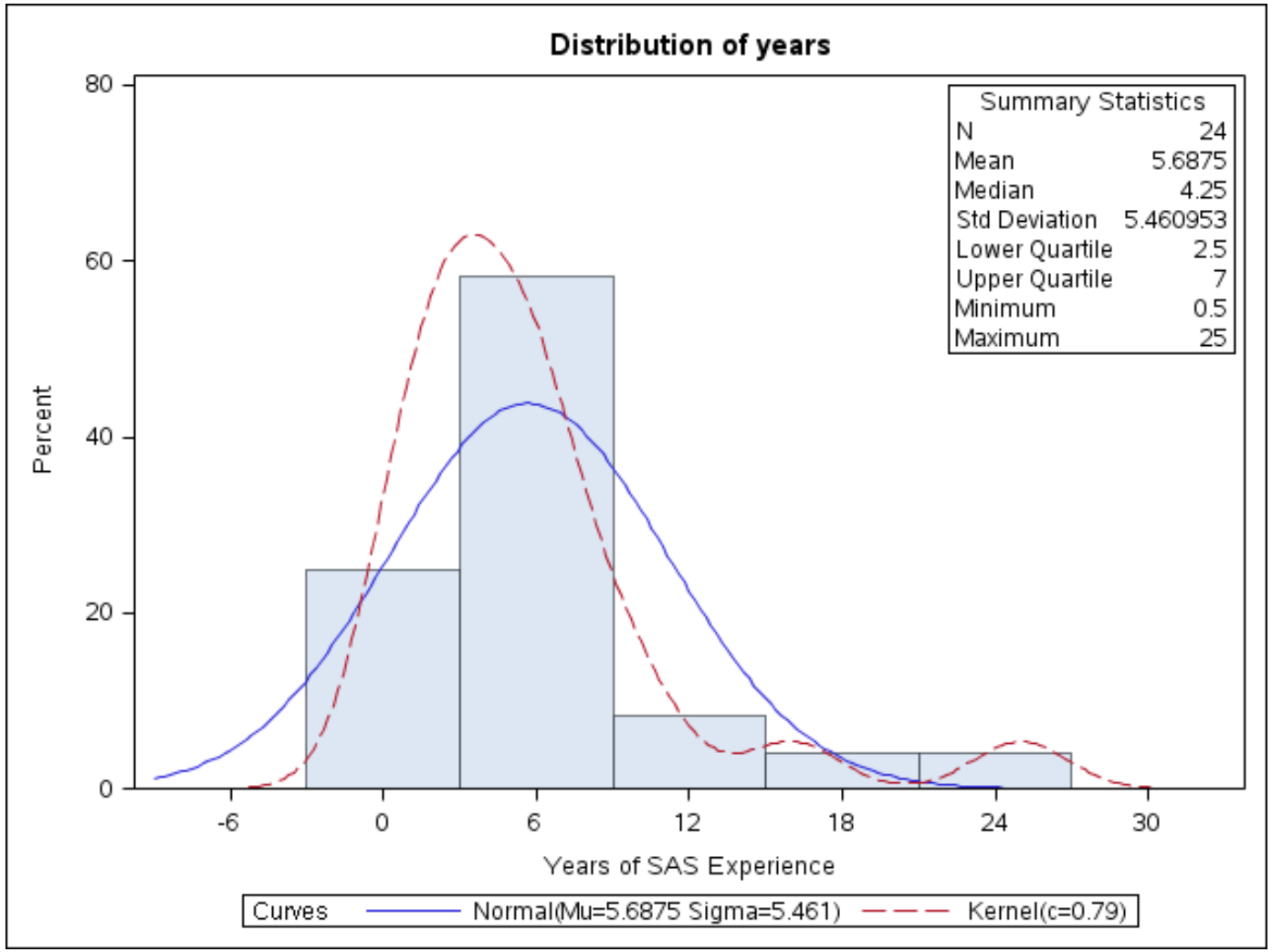
```
options nocenter mprint symbolgen fullstimer;

%let ds=adv_sas;
%let var=years;
%let var_label = Years of SAS Experience;

data &ds.;
  input &var @@;
  datalines;
  3 1.5 .5 6 5 7 7 3.5 3.5 10 5.5 16 7 .5 5
  2 1 6.5 3.5 25 10 3 1 3.5
  ;;
run;
```

MACROS in SAS. MACRO VARIABLE ASSIGNMENT

- Continuation of code:

```
ods graphics on;
proc univariate data=&ds. ;
   var &var.;
   histogram &var.   / vscale = percent
                       normal(color=red)
                       kernel(color=blue)
                       ;
   inset n mean  median  std q1 q3 min max
         /header = 'Summary Statistics'
          pos=ne
          ;
  label years = "&var_label. ";
run;
```

MACROS in SAS. MACRO VARIABLE ASSIGNMENT



Example 3: Same output as in 2 but within a MACRO

MACROS in SAS. More Advanced MACRO CODE

```
%macro uni_hist(data_set,var,
                var_label,
                where,format,density,stats,pos);
ods graphics on;
proc univariate data=&data_set. ;
   var &var.;
   &where;
   histogram &var.   / vscale = percent
                &density
                ;
   inset &stats
         /header = 'Summary Statistics'
          pos=&pos
          ;
   label &var. = "&var_label."
run;
%mend;
```

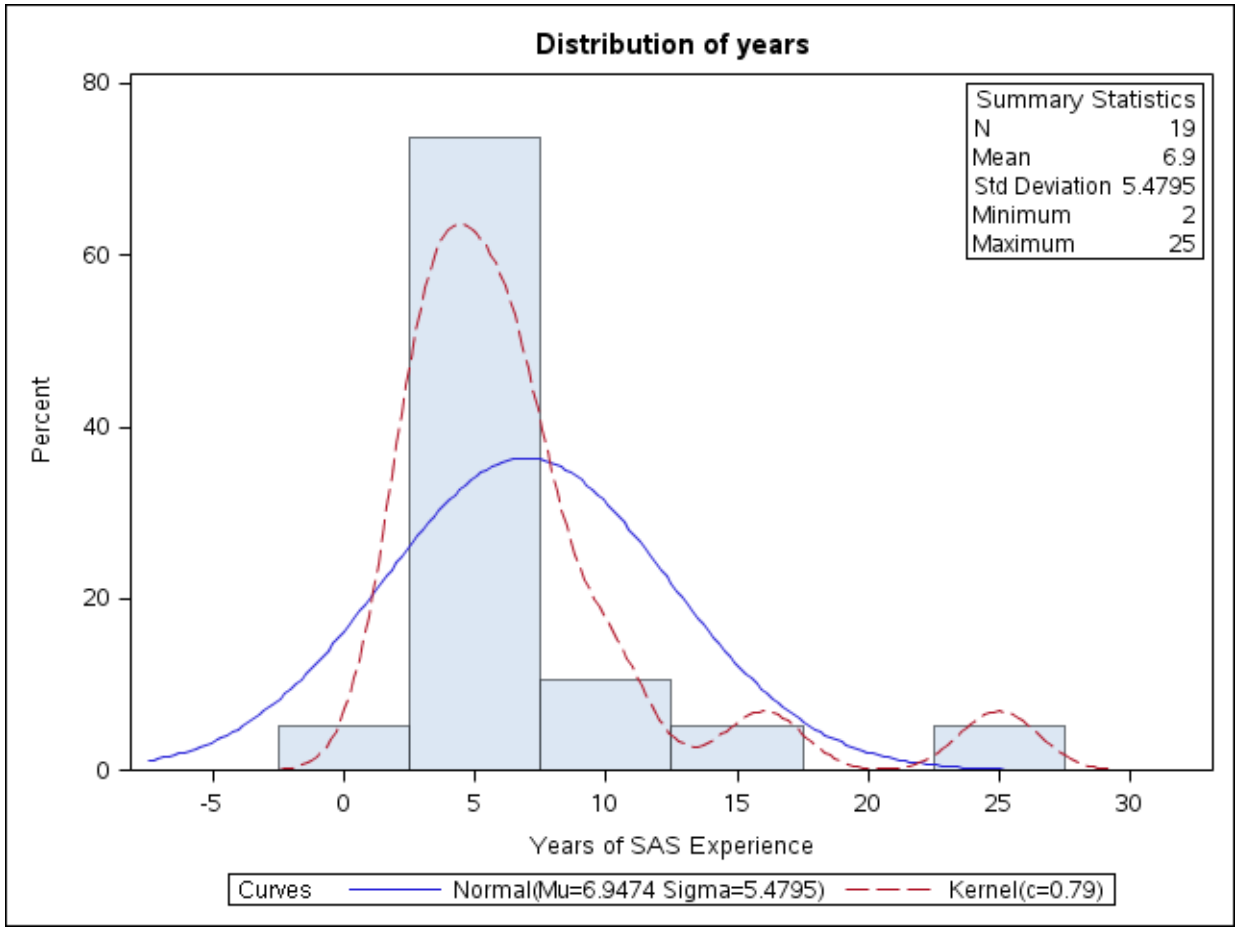
MACROS in SAS. More Advanced MACRO CODE

```
/*  MACRO CALL */
%uni_hist(data_set=adv_sas,
        var=years,
                var_label=Years of SAS Experience,
        where=%STR(where &var. >=2;),
                density=normal kernel,
        stats= n mean(5.1) std(8.4) min max,
        pos=NE,
        var_label=Years of SAS Experience
        );
```

NOTES:

- **You may ned to add SPOOL or NOSPOOL OPTION to the OPTIONS statement. This maybe recommended in the LOG .**

More Advanced MACRO CODE.



Excluding the where clause

```
/*  MACRO CALL */
%uni_hist(data_set=adv_sas,
        var=years,
        where=,
                density=normal kernel,
        stats= n mean(5.1) std(8.4) min max,
        pos=NE,
        var_label=Years of SAS Experience
        );
```

```
/*  MACRO CALL */
%uni_hist(adv_sas,
        years,
                ,
                normal kernel,
        n mean(5.1) std(8.4) min max,
        NE,
       Years of SAS Experience
        );
```

More TRIVIA QUESTIONS?

TRIVIA QUESTION &q.

- **How can you comment out a %LET statement? More than 1 answer.**
  a. ***%let var_label = Years of SAS Experience;**
  b. **%*let var_label = Years of SAS Experience;**
  c. **/* %let var_label = Years of SAS Experience; */**
  d. **%macro hold; %let var_label = Years of SAS Experience; %mend;**

TRIVIA QUESTION

- Small LEGO MACRO. Anything wrong with it?

```
%macro TL(KEY,FMT);
  input(put(&KEY.,&FMT..),best32.)
%mend;
```

TRIVIA QUESTION

- Small LEGO MACRO. Anything wrong with it?

```
%macro TL(KEY,FMT);
  input(put(&KEY.,&FMT..),best32.)
%mend;
```

USAGE in OPEN CODE or OTHER MACROS:

```
SCORE = 5  +
      10 * %TL(RUN_DATE,UNEMPLOTMENT_RATE) +
      12 * %TL(SEG,AVG_LOSS_ASSUMPTION);
```

CALL SYMPUTX
or
CALL SYMPUT

USING CALL SYMPUTX within a DATA STEP

- CALL SYMPUTX is a useful way to assign values to macro variables within a data step. Once assigned the macro variables can be used in the program.
- Syntax:
  - call symputx(“macro-variable”, value);
    - macro-variable can be with quotes naming the macro variable or a character variable value if not quoted.
    - Value can be character or numeric value or a data set variable.
- **Macro variables get assigned after the data set is run. Cannot use the macro variable in the data step.**
- CALL SYMPUTX replaces CALL SYMPUT since CALL SYMPUTX eliminates leading and lagging spaces in both the MACRO variable and the value.

USING CALL SYMPUTX within a DATA STEP

Thanks David B. Horvath, NOBS for Noobs. PHILASUG FALL 2015

```
options nocenter;

data _null_;
  call symputx('nobs',n);
  set sashelp.cars nobs=n;
  stop;
run;
%put nobs = &nobs;
```

This code only works with SAS Data Sets.

```
%put nobs = &nobs;
nobs = 428
```

Capturing SAS statistics into Macro Variables using ODS OUTPUT

Will Hold for a Rainy DAY.

Involves a few steps:

- **ODS TRACE ON/LISTING**
- **ODS TRACE OFF;**
- **ODS OUTPUT**
- **ODD OUTPUT CLOSE**
- **DATA _NULL_;**
- **CALL SYMPUTX**

Few MACRO Functions

A few MACRO Functions: Quoting

%STR:

```
%let task = %str(proc print data=_last_ noobs;
                  var _numeric_;
                  format _numeric_ best32.;
                 run;);
```

In open code or macro code, &task will run the whole PROC.

%BQUOTE is another way to include multi line or strings with open quotes.

A Few MACRO Functions: %SCAN

Problem: Capitalize all variable names in a data set.

TEST DATA:

```
data test;
  AAA = 5;   bbb =6;  CC = 7;  ddd = 8; xYz='BB';
  output;
run;  /* How many observations in test? */

proc contents data=test noprint out=cnts; run;
/* Why are we creating an output data set from PROC CONTENTS? */
```

Why are we creating an output data set from PROC CONTENTS?

- Output data from PROC CONTENTS provides a data set that has information about the data and the variables. CNTS data has 5 observations and 40 variables.

- Variables:

  CHARSET COLLATE COMPRESS CRDATE DELOBS ENCRYPT ENGINE FLAGS FORMAT FORMATD
  FORMATL GENMAX GENNEXT GENNUM IDXCOUNT IDXUSAGE INFORMAT
  INFORMD INFORML JUST LABEL LENGTH LIBNAME MEMLABEL MEMNAME MEMTYPE MODATE NAME
  NOBS NODUPKEY NODUPREC NPOS POINTOBS PROTECT REUSE
  SORTED SORTEDBY TYPE TYPEMEM VARNUM

- Some of the Variable values:

Obs	NAME	TYPE	NOBS
1	AAA	1	1
2	CCC	1	1
3	bbb	1	1
4	ddd	1	1
5	xYz	2	1

- TYPE: 1=numeric 2=character

Back to our code:

```
proc sql noprint;
 select name into: vars separated by ' '
    from cnts
 ;
quit;
/* The above code generates a macro variable called
vars that has each variable name separated with a
space */

%put vars = &vars.;
```

LOG OUTPUT:
72 %put vars = &vars.;
vars = AAA CCC bbb ddd xYz
73

The %SCAN function:

%SCAN pulls out parts of a macro variable.

- The function has 3 arguments:
  - First argument is the macro variable name.
  - Second argument is the part number. 1=first part, 2= second.
  - The 3rd argument defines the delimiter that separates the parts of the macro variable. This example has the delimiter as a space. Could be a comma.

The MACRO:

```
%macro rename;
%let I = 1;                                        /* Initialize &I. */

proc datasets lib=work nolist;                     /* DATASETS to modify the data*/
 modify test;


 %do %until(%scan(&vars,&I.,%str( ) ) = %str( ) );
   %let var=%scan(&vars,&I.,%str( ) );
   %let ren=%upcase(&var.);                        /* &REN=Upcase of &var */
   %if &var. ne &ren %then %do;                    /* if &VAR not equal to &REN*/
     rename &var. = &ren.;                         /* Use RENAME STATEMENT in PROC DATASETS */
   %end;                                           /* END THE 2nd LOOP */
   %let I = %eval(&I. + 1);                        /*  Increment &I by 1 */
 %end;                                             /*  End First Loop */
 run;quit;

proc contents data=test;
run;
%mend;                                             /*  END MACRO */

%rename                                            /*  CALL MACRO  */
```

Thanks, Michael A. Ratithel *PROC DATASETS; The Swiss Army Knife of SAS® Procedures*
Michael A. Raithel, *Westat, Rockville, MD and SAS Press Author*

Contents Result:

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
1	AAA	Num	8
2	BBB	Num	8
3	CCC	Num	8
4	DDD	Num	8
5	XYZ	Char	2

%EVAL or %SYSEVALF?

A Few MACRO Functions: %EVAL

Performs integer arithmetic on macro number variables. Results are integer even if the arithmetic result is not an integer.

```
1       options nocenter;
2
3       %let x=5;
4       %let y=&x.+1;
5       %let z=%eval(&x.+1);
6       %let a=%eval(&x/&y);
7
8       %put &x &y &z &a;
5 5+1 6 2
```

IF I had
%let a=%eval(&x/&z)
I will get 0 printed out.

```
%let a=%sysevalf(&x/&z);  /* SOLUTION is SYSEVALF */
 will print out
0.83333333333333
```

SYSTEM MACRO VARIABLES

SYSTEM MACRO VARIABLES:

%put _all_; /* will list all macro variables in the log: AUTOMATIC, GLOBAL, LOCAL */

- Results from %put _all:
  - A large list of macro variables set up automatically when you start SAS and or run SAS code that contains user defined macro variables.
- There maybe a large number of AUTOMATIC variables listed. Could be confusing but take a look at it. You may find something interesting that you can use in your code. As Arte Johnson said on Rowan & Martin Laugh-In: “Verrry interesting…”

SYSTEM MACRO VARIABLES

A few examples will be shown on the next 7 pages.

Get FUNKY with %SYSFUNC

Example of using SYSTEM MACRO VARIABLES

- Adding System Date and Time to title of an ODS output file:

```
ods EXCEL file = "MODEL_3YR.&sysdate..%sysfunc(compress(&systime,':')).xlsx"
  style=SASWEB;
```

- Double quotes required for macro variable resolution.
- &: Indicates a macro variable.
- .: Indicates the end of the macro variable. Sometimes you need this sometimes you don't.
- %: Indicates a macro call, macro statement, or a macro function. This case it is a SYSFUNC function used to apply some data step functions to macro variables.
- Example output: MODEL_3YR.04JUN15.1837.xlsx
- WILL WORK IN open code.

Moriak, C. "%SYSFUNC: A Macro Variable Can't Function Without It" NESUG 15

Example of using SYSTEM MACRO VARIABLES

- NOTE: When running interactively, e.g. SAS/EG certain rules may apply:
  - &SYSTIME will not change, unlike batch submit, until you exit SAS/EG.
  - You will need to specify the full path of the location of the ODS output.
  - See next page to see an example run in SAS/EG

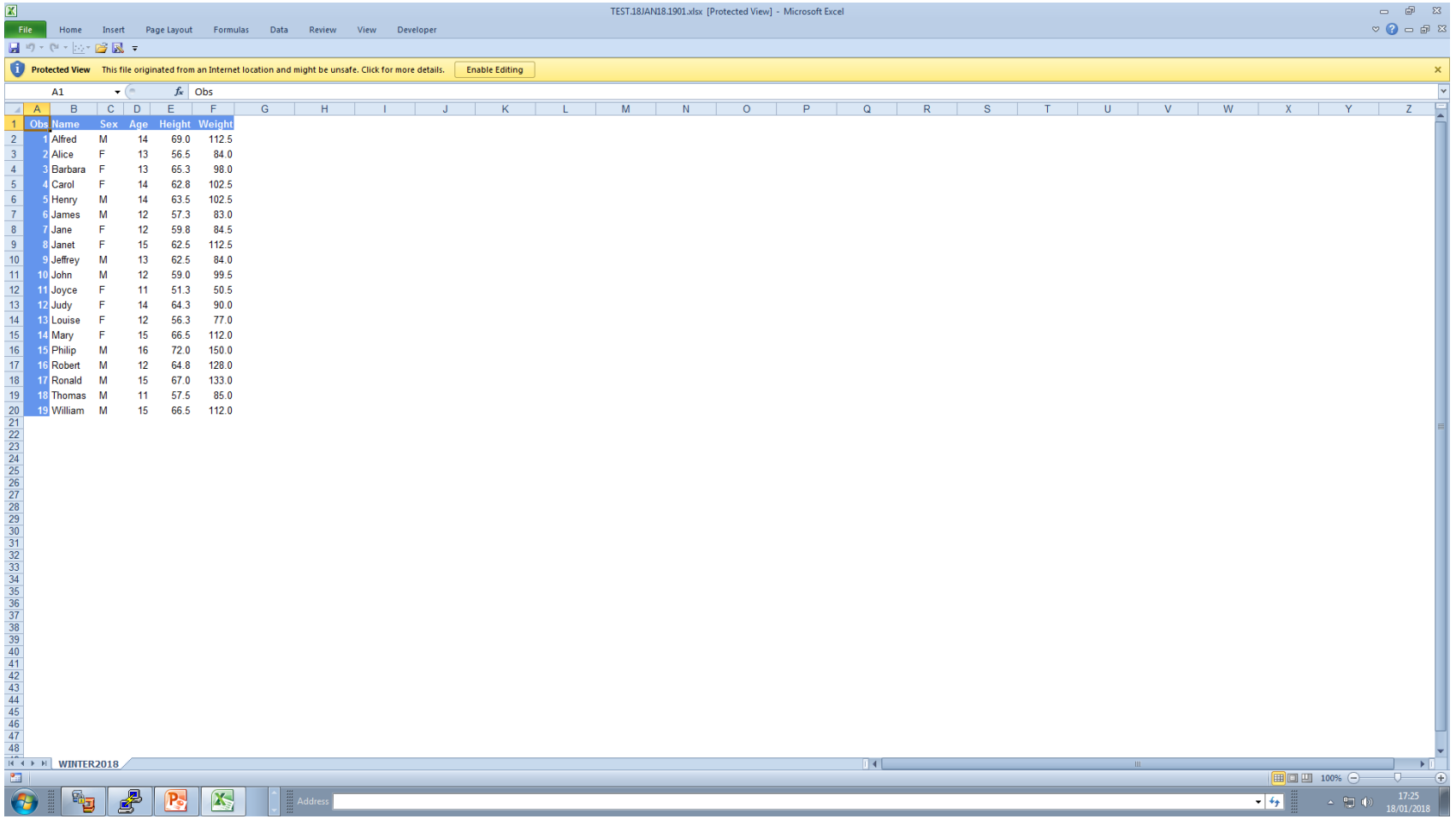
DUMPING ODS OUTPUT TO your WORK LOCATION

- Running SAS/EG on a GRID you can also see what is your work location.
  - Global variable &SASWORKLOCATION.
- Why dump to work location?
  - Your output will be available for FTP from the work location to your LAN location after program runs if using SAS/EG.
    - Not so for batch submit code.
  - If you exit SAS/EG correctly, all work space files will be deleted including the XLSX file.
  - No need for manual clean up of files.

DUMPING ODS OUTPUT TO your WORK LOCATION

```
%let outp=%sysfunc(compress(&SASWORKLOCATION.,'"'));


ods EXCEL  file="&outp.TEST.&sysdate..%sysfunc(compress(&systime,':')).xlsx“
                style=SASWEB;
ods EXCEL  options(sheet_name="WINTER2018");

proc print data=sashelp.class;
run;

ods EXCEL close;
```

DUMPING ODS OUTPUT TO your WORK LOCATION



Looping in a MACRO CODE

Fruit Loops

```
1           options nocenter ;
2           %macro loopy_run;
3             %do dog = 1 %to 4;
4               %put dog = &dog;
5             %end;
6           %mend;
7
8           %loopy_run;
dog = 1
dog = 2
dog = 3
dog = 4
```

Fruity Loops

```
1          options nocenter ;
2          %macro loopy_run;
3            %do dog = 1,2,3,4;
ERROR: Expected %TO not found in %DO statement.  A dummy macro
will be compiled.
4              %put dog = &dog;
5            %end;
6          %mend;
7
8          %loopy_run;

           _
           180
WARNING: Apparent invocation of macro LOOPY_RUN not resolved.
```

More Loops?

```
1           options nocenter ;
2           %macro loopy_run;
3             %do dog = Poodle %to Lab;
4               %put dog = &dog;
5             %end;
6           %mend;
7
8           %loopy_run;
ERROR: A character operand was found in the %EVAL function or %IF
 condition where a numeric operand is required. The condition was:
       Poodle
ERROR: The %FROM value of the %DO DOG loop is invalid.
ERROR: A character operand was found in the %EVAL function or %IF
 condition where a numeric operand is required. The condition was:
       Lab

ERROR: The %TO value of the %DO DOG loop is invalid.
ERROR: The macro LOOPY_RUN will stop executing.
```

More Loops

```
options nocenter mprint symbolgen fullstimer;
proc format;
  value doggy  1 = 'Poodle'
               2 = 'Basset Hound'
               3 = 'Lab'
  ;
run;

%macro loopy_run;
  %do dog = 1 %to 3;
    %let breed = %sysfunc(putn(&dog.,doggy.));
    %put dog = &dog  | breed = &breed.;
  %end;
%mend;

%loopy_run;
```

LOG:

```
dog = 1  | breed = Poodle
dog = 2  | breed = Basset Hound
dog = 3  | breed = Lab
```

Storing Macros in Folders to Call From Other Source Codes

AUTOCALL Facility

You can save macros in a directory for SAS to look for the macros when called.

options mautosource **SASAUTOS=(mac1 mac2);**

You can refer to more than 1 location:

filename mac1 “\stuff\models\macros\”;
filename mac2 “\stuff\macros”;

This may not work in the SAS GRID. To work you will need to move macro programs from the LAN to the GRID.

Compiling MACROS

You can save macros in a compiled catalog in a directory for SAS to look for the macros when called. Example from NPV simulation code:

```
options nocenter mprint symbolgen compress=binary fullstimer
      mstored sasmstore=macin source2;

libname macin ('.');
%macro PREPAYMENT/ store source des='PREPAYMENT MODELS';
..
%mend PREPAYMENT;
```

Note that the source code is not recoverable from the macro catalog unless you specify a SOURCE option.

libname macin ('.'); If you are running SAS interactively or SAS/EG or the GRID you will need to provide the full path name.

ALSO NOTE: CATALOGS cannot be FTPed across different platforms

Compiled MACROS

You can store catalogs in multiple directories:

```
options nocenter mprint symbolgen compress=binary fullstimer mstored
        sasmstore=macin source2;

libname macin ('../../MACROS', './MODELS') access=readonly;
```

Above libname concatenates 2 catalogs where the compiled macros reside. Note, if running interactively, you will need to provide the full path. Above code is for UNIX directory commands running a batch SAS program.

Recovering code of compiling MACROS

```
%copy prepayment / lib= macin
     out='c:\macros\prepayment.sas' src;
```

%copy is a system macro provided by SAS.

Note that the source code is not recoverable from the macro catalog unless you specify a SOURCE option when compiling.

Listing of MACRO Catalog

```
proc catalog catalog=macin.sasmacr;
  contents /*stats*/;
run; quit;
```

Contents of Catalog MACIN.SASMACR

#	Name	Type	Create Date	Modified Date	Description
1	BLEND	MACRO	11Apr16:18:55:11	11Apr16:18:55:11	Macro to blend some fields in MTG NPV
2	CAP_REPAY	MACRO	11Jul16:15:38:46	11Jul16:15:38:47	CAP_REPAY
3	FIND_MISSING	MACRO	11Apr16:17:57:34	11Apr16:17:57:34	Finds missing value variables
4	FTP_CALC	MACRO	19Apr16:20:05:10	19Apr16:20:05:10	FTP calc:FTP_LTP_RATE
5	FTP_LIBOR	MACRO	11Apr16:18:24:46	11Apr16:18:24:46	FTP calc:FTP_Rates_DIV_LIBOR_RATE
6	INPUT_FILE	MACRO	14Jul16:19:10:54	14Jul16:19:10:54	Captures INPUT_FILE
7	K_CALC	MACRO	14Apr16:19:55:20	14Apr16:19:55:20	calculates K
8	MNTH	MACRO	01Jun16:14:07:14	01Jun16:14:07:14	Adds Month Header
9	OBS_VARS	MACRO	01Jun16:13:16:13	01Jun16:13:16:13	Returns NOBS NVARS macro variables
10	PARMREAD	MACRO	11Apr16:17:51:09	11Apr16:17:51:09	Pulls in PARMs from &INPUT_FILE..xls
11	PRINT10Y	MACRO	12Jul16:13:42:51	12Jul16:13:42:51	10 YR Macro Print P&L.V2
12	PRINT10_ODS	MACRO	22Apr16:18:39:18	22Apr16:18:39:18	10 YR ODS Print P&L.V2
13	SOURCE_CODE	MACRO	12Apr16:16:41:18	12Apr16:16:41:18	Prints Full Source Code Name
14	YEAR6P	MACRO	11Apr16:18:12:13	11Apr16:18:12:13	IF ONLY 5 years if INPUTS provided
15	YEARLY_OUT	MACRO	12Apr16:16:52:20	12Apr16:16:52:20	Outputs Yearly metrics

Deleting entries of a MACRO Catalog

```
proc catalog catalog=macin.sasmacr ET=MACRO;
  delete PRINT10_ODS
        /* You can add more than 1 entry *
  ;
run;
  contents /*stats*/;
run; quit;
```

Odds and Ends

Debugging SAS MACROS

Pain to debug MACROS.
The macro is often treated in the log as a single line of code. You get errors, for example, pointing to line 156, column 456. Line 156 is the start of the macro with errors but it is hard to determine column 456.

MFILE option is the solution.

MFILE: Output resulting source code to a file for debugging

```
options MPRINT MFILE;
filename mprint ‘c:/sasmacros/sascode.sas’
```

To turn off option:

```
options MPRINT NOMFILE;
*filename mprint ‘c:/sasmacros/sascode.sas’
```

MFILE generates code that can replace the macro to test in a second run with an %include MPRINT as opposed to calling the MACRO.
Once you generate the log with that code as opposed to the macro call, errors will be better trcked.
Other options are available for debugging. SEE:
SAS® System Options: The True Heroes of Macro Debugging Kevin Russell and Russ Tyndall, SAS Global Forum 2010
https://support.sas.com/resources/papers/proceedings10/147-2010.pdf

MACRO code Syntax: Using macro to comment out code. Just don’t call the macro.

```
%macro hold;
proc print data=/*sashelp.*/class;
run;
%mend;
```

MACRO code with Big Data:

MACRO logic was introduced years ago when storage was more expensive and more of an issue. If MACROS are not used properly, they can slow down processing.
Log files can be large. Once a macro is tested and compiled, make sure these options are coded: NOMLOGIC, NOMPRINT, NOMRECALL, NOSYMBOLGEN. See:
Big Data, Fast Processing Speeds Kevin McGowan SAS® Solutions ...
support.sas.com/resources/papers/proceedings13/036-2013.pdf

Questions?

Comments?

Concerns?