CMU MSP 36-602: SAS Macros I

Howard Seltman, April 1, 2019

1) **Why learn SAS macros?**
   a. Many jobs list it as required or desired
   b. It is an example of a useful computing technique you probably haven't seen: programs that write programs

2) **Macro Quickstart Examples**
   a. Setting "constants"

```
%LET cutoff = 2;
... a bunch of code
PROC GLM DATA=myData(where=(myVar >= &cutoff));
  CLASS myCat;
  MODEL myY = myCat myQuant / SOLUTION;
RUN;
... even more code
```

   b. A "function"

```
/* Define the macro */
%MACRO myMacro(cutoff, data=myData, var=myVar);
  ... a bunch of code
  PROC GLM DATA=&data(WHERE=(&var>=&cutoff));
    CLASS myCat;
    MODEL myY = myCat myQuant / SOLUTION;
  RUN;
  ... even more code
%MEND myMacro;
/* Somewhere else, run the macro */
%myMacro(cuoff=5, var=myOtherVar)  /* defaults to myData */
```

3) **Macro Concepts**
   a. The **uses** of the macro facility are to
      i. pass information to or between DATA and/or PROC steps (macro variables)
      ii. automate complex and/or repetitive tasks, similar to an R function (macros)
      iii. simulate the use of FOR, IF, etc. for PROCs (macros)

   b. **Why** can't we do these directly, like in R?
      i. Data are stored in files not memory (more efficient for large data)
      ii. PROCs are compiled, self-contained programs (faster than in R)

c. ➔The macro facility's two basic **capabilities** are **storing/retrieving data values** and **dynamically generating code**. The latter is complemented by 1) special functions that connect DATA steps to the macro system and 2) ODS OUTPUT with PROC-specific "ODS table names" which sends PROC output to SAS datasets.

d. The **macro language** is a set of additional syntactic elements that you put into your SAS code. For macros per se (i.e., macro "functions"), the **macro processor** compiles your macro code into a macro, which is a text generator that writes SAS code (one SAS step at a time), and then passes it on to the usual SAS compiler. Therefore, ***the macro facility can have no effects during DATA or PROC execution*** (with some very specific exceptions).

e. The ampersand ("&") and percent ("%") symbols, plus certain DATA step functions, e.g., `SYMGET( )`, and calls, e.g., `CALL SYMPUT( )`, are **indicators of the use of the macro language**. The "&"symbol in front of a macro variable name tells the macro facility to insert the value of the variable into the code. The "%" symbol is a prefix for **macro statement keywords** (including the keywords `%MACRO` and `%MEND` which bracket the definition of a macro per se), for **macro-specific functions**, and for **invoking a user-defined macro** (i.e., running it).

f. **Exception**: The following starts with `%`, but is not part of the macro system:
`%INCLUDE myFileName;` (to load and run code from a file)

g. **Alternatives** to use of macros include
   i. passing information solely in the form of data sets (ODS)
   ii. use of the Interactive Matrix Library (advanced)
   iii. use of the SAS Toolkit to write new PROCs in C (very advanced)

4) Macro Variables

a. Macro variables are **named strings that persist** across DATA and PROC steps within a SAS session. Names can be up to 32 characters and can contain letters, underscores and digits (not in the first position). Macro variables with names that match dataset names, variable names, LIBNAME's, or FILENAME's are distinct from them. "Local" macro variables are distinct from global macro variables with the same name (see below). The names are ***not*** case-sensitive.

b. **Macro variable creation** (allowed almost anywhere, but pointless in DATA/PROC steps):
Syntax: `%LET myVar = myUnquotedString;`
The `%LET` assignment statement is allowed in "open code", not just inside macros. Everything between the first non-blank character after the equals sign and the last non-blank character before the semicolon ends up in the variable, including any quotes.

c. **Syntax for macro variable use** (allowed almost anywhere, but ***not*** expanded inside of ***single*** quotation marks or the %NRSTR( ) function): &myVar or &myVar.

Wherever either &myVar or &myVar. is found, your SAS code is re-written by "expanding" the reference into the actual macro text corresponding to the macro variable. After this "text preprocessing" the code is run in the usual way.

The period form is needed, e.g., if you want to include letters immediately following the substituted text:

```
%LET hydrogens = 2;
PROC MEANS DATA=water;
  VAR H&hydrogens.O;
RUN;
```

This code will be converted to:

```
PROC MEANS DATA=water;
  VAR H2O;
RUN;
```

The result is calculation of the mean of the H20 variable in the water dataset. But if the period were missing, the SAS Macro Processor would try to look up the macro variable "&hydrogensO" , which would result in a message like "WARNING: Apparent symbolic reference HYDROGENSO not resolved."

d. **Special macro statement: %PUT**

Syntax: %PUT [space-separated-text and/or &var and/or macro-%functions];
The %PUT statement prints text, possibly including the contents of some macro variables, to the SAS Log. It is allowed in "open code", including DATA steps and PROC steps (pointless), as well as inside macros.

i. Example: %PUT myVar equals &myVar;
ii. Example: %PUT myVar starts with %SUBSTR(&myVar,1,1);
iii. Also, %PUT _USER_; shows all of your macro variable definitions.

e. **Special macro statement: %SYMDEL**
Syntax: %SYMDEL myVar1 [myVar2 [myVar3…]];
Deletes macro variables. Allowed in "open code", not just in macros.

i. Example: %SYMDEL myVar1;
ii. Example: %SYMDEL tax period;

f. Note that macro variables ***always*** begin with an "&" ***except*** immediately after the keywords %LET or %SYMDEL or in functions like %SYMEXIST( ) or %SYMLOCAL( ). The exceptions are because macro variable names are the ***only*** things that can appear in those places. Also, code like %SYMDEL &varToKill; is useful.

g. **Simple, but useful, example:**

```
%LET currentYear = 2017;
LIBNAME fiscal "fiscalData";
TITLE "Report for &currentYear";
PROC FREQ DATA=fiscal.FY&currentYear;
  TABLES expenses;
RUN;
PROC MEANS DATA=fiscal.FY&currentYear;
  VAR amtSpent;
RUN;
```

h. Extended example:

```
/* Goal: Add monthly expenses to full expense data set,
   and report year-to-date results. */

/* Requires: Text file "ExpensesMMMYYYY.dat" must be in
   the current directory with variables "category" and "amount".
   Permanent dataset "expenses" is in the "mylib" folder. */

/* Change just these two lines each month */
%LET year = 2017;
%LET month = Feb;

FILENAME infile "Expenses&month&year..dat";
LIBNAME mylib "mylib";

DATA E&month&year; /* temporary dataset */
  INFILE infile;
  LENGTH category $12. month $3.;
  INPUT category amount;
  year = &year;
  month = "&month";
RUN;

DATA mylib.expenses;
  SET mylib.expenses E&month&year;
RUN;

TITLE "Year to date up to &month &year";
PROC MEANS DATA=mylib.expenses (WHERE=(year=&year));
RUN;
```

i. **Technical details for %LET**

   i. No quotes are needed. If quotes are included, they end up in the variable.

   ii. Digits are treated as text, i.e., ***everything is a string***. The same goes for arithmetic expressions, i.e., the whole expression ends up in the string.

   iii. You can include %EVAL(non-decimal expression) or %SYSEVALF(any expression) to put the text corresponding to an expression's result into the macro variable:

```
%LET age1 = 2019 - 1954;
%LET age2 = %EVAL(2019 - 1954);
%PUT age1 = &age1 and age2 = &age2;
```

   Log shows: `age1 = 2019 - 1954 and age2 = 65`

   iv. Aside: Using SAS as a calculator: `%PUT %SYSEVALF(3.5 + (2.54*6)/3);`

   v. A "null" value is allowed: `%LET country =;`

   vi. & and % are evaluated during the %LET assignment. %STR( ) can be used to "protect" these.

```
%LET name1=Barnes;
%LET name2=Noble;
%LET company=&name1 and &name2;
%PUT &Company;
```

   In the Log: `Barnes and Noble`

```
%LET company=S&H Green Stamps;
```

   In the Log: `WARNING: Apparent symbolic reference H not resolved.`

```
%LET company=S%STR(&)H Green Stamps;
%PUT &Company;
```

   In the Log: `S&H Green Stamps`

vii. %LET discards leading and trailing spaces and ends with the semicolon (ignoring semicolons inside paired quotes). To include code with a semicolon, use **%STR()** to convert its entire contents to a string.

```
%PUT "use proc print data=dat;";
In the Log: "use proc print data=dat;"

%LET doit = PROC MEANS; VAR x; RUN;
                        ---
                        180
ERROR 180-322: Statement is not valid or it is used out of
proper order.

%PUT &doit;      shows PROC MEANS in the log.

%LET doit = "PROC MEANS; VAR x; RUN;";  /* no error */
&doit
NOTE: Line generated by the macro variable "DOIT".
"PROC MEANS; VAR x; RUN;"
------------------------
180
ERROR 180-322: Statement is not valid or it is used out of
proper order.

%LET doit = %STR(
  PROC MEANS;
  VAR x;
  RUN;);   /* also works on one line */
&doit
NOTE: There were 4 observations read from the data set WORK.FAKE.
NOTE: PROCEDURE MEANS used (Total process time):
real time           0.01 seconds
cpu time            0.01 seconds
```

viii. A % can be used to "protect" an unmatched single or double quote or a percent sign or unmatched parenthesis inside of %STR():

```
%PUT Favorite punctuation: %STR(%%, %'s, %).);
Favorite punctuation: %, 's, ).
```

ix. %LET can include substitution of macro text: `%LET myCode=%myMacro;`

x. A period after a macro variable is a "disappearing" delimiter, silently indicating the end of the macro variable (see *4c* above).

xi. The equivalent of %LET, while actually running inside a DATA step, is the `CALL SYMPUT()` routine, which assigns a string to a macro variable. Note that there is no % or & involved, and you usually will want to remove extra spaces.

```
DATA fruit;
  %LET stupidDemo = x$ y; /* in/outside: precedes DATA */
  INPUT &stupidDemo @@;
  DATALINES;
  Apple 3 Pear 4 Grapes 10 Mango 1
RUN;
DATA fruit;
  SET fruit END=lastLine;
  IF lastLine THEN DO;
    CALL SYMPUT('lastx', TRIM(x));
    CALL SYMPUT('lasty', TRIM(LEFT(PUT(y,8.))));
  END;
RUN;
%PUT Last count is &lasty and &lastx is the last fruit.;
```

In the Log: `Last count is 1 and Mango is the last fruit.`

Important note: the macro variables "lastx" and "lasty" are not available for use until ***after*** the DATA step ends, i.e., they cannot be used to carry information within a DATA step (use RETAINed variables instead).

Easier version: `CALL SYMPUTX('lasty', y);` converts, trims, and justifies

5) **Macro function examples**:

a. %LENGTH()

i. Syntax: `%LENGTH(argument)`
ii. Action: returns the number of characters in the argument
iii. E.g., `%LENGTH(&stupidDemo)` (using "stupidDemo" from above) is replaced by "4" in your code.

b. %SUBSTR()

i. Syntax: `%SUBSTR(argument, position[,length])`
ii. Action: returns the substring of "argument" starting at "position" and continuing to the end or only "length" characters long.
iii. E.g., `%SUBSTR(&doit, 18, 5) (using "doit" from above)` is replaced by "`VAR x`" in your code.

c. %EVAL() Note: compare to `%SYSEVALF()` which does "floating point"

i. Syntax: `%EVAL(argument)`
ii. Action: returns the value for an integer mathematical expression
iii. E.g., `%EVAL(101/25+3)` is replaced by "7" in your code and `%EVAL(1/3)` is replaced by "0" in your code.

d. %SCAN()

i. Syntax: %SCAN(text, index, delimiter)

ii. Action: return the index'th word from "text" using the "delimiter"

iii. `%PUT %SCAN(&SYSDSN,1,%STR( ));` returns the LIBNAME of the most recently used dataset. Index 2 gives the dataset name.

iv. The default delimiters are blank ! $ % & ( ) * + , - . / ; < ^

e. For others, search "macro functions" (with the quotes) in SAS Help.

f. Silly, informative, example:

```
DATA prepost;
  INPUT preQ1 preQ2 preQ3 preQ4
        postQ1 postQ2 postQ3 postQ4
            preH1 preH2 postH1 postH2;
  DATALINES;
  20 30 40 50  21 31 41 51  50 90 52 92
  11 22 33 44  16 26 36 46  33 77 42 82
  9 8 7 6      19 18 17 16  17 13 37 33
RUN;

%LET tax = pre;
%LET period = Half2;

%LET var =
   &tax%SUBSTR(&period,1,1)%SUBSTR(&period,%LENGTH(&period));
TITLE "Taxes for &var";
PROC MEANS DATA=prepost;
  VAR &var;
RUN;
```

g. Example for similar, but different datasets (different numbers of columns)

```
%LET fname = experiment3.dat;
%LET count = 4;  /* varies by experiment */

DATA exper;
  ARRAY s{&count};
  ARRAY a{&count};
  INFILE "&fname" FIRSTOBS=2;
  INPUT id treatment handicap t1-t&count s1-s&count;
  DO i = 1 to &count;
    a[i] = s[i] - handicap;
  END;
  DROP i;
RUN;
PROC PRINT DATA=exper;
RUN;
```