

Writing cleaner and more powerful SAS code using macros

Patrick Breheny

Why Use Macros

- Macros automatically generate SAS code
- Macros allow you to make more dynamic, complex, and generalizable SAS programs
- Macros can greatly reduce the effort required to read and write SAS Code

Outline

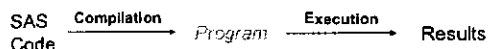
1. Quick Review of Macro Basics
2. Working with Macro Strings
3. Getting More Out of Macros:
 - a) Program Control
 - b) Interfacing with Data

The Compilation of SAS Programs

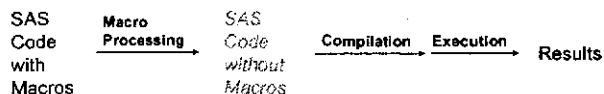
- SAS code is compiled and executed alternately in steps:
 - For example, a data step will be compiled and executed, then a procedure step will be compiled and executed
- **IMPORTANT:** Macros are resolved **PRIOR** to the compilation and execution of the SAS code

SAS Compilation (cont'd)

- Code without Macros:



- Code with Macros:



Macro Code

- The two basic elements of macro code are *macro variables* and *macros*. In SAS code:
 - *&name* refers to a macro variable
 - *%name* refers to a macro
- Macro code consists of these two elements and their relationship to each other
- Text without %'s or &'s (called *constant text*) is unaffected by macro processing

Working with Macro Strings

Concatenation

- The expression `survey_&state` is unambiguous, but what about `&state_survey`?

```
%put survey_&state;
survey_IA

%put &state_survey;
WARNING: Apparent symbolic reference
STATE_SURVEY not resolved.
&state_survey
```

- A period is the signal in SAS to end a macro variable name:

```
%put &state._survey;
IA_survey
```

Double vs. Single Quotes

- Double quotes and single quotes affect macro variables differently:

```
proc import datafile='H:\Macro Workshop\survey_&state..xls'
out=survey_&state
replace;

run;

ERROR: Unable to import, file
H:\Macro Workshop\survey_&state..xls does not exist.
```

- Note that macro variables inside single quotes are not resolved

The Implicit Handling of Strings

- Because macros and macro variables can only be assigned strings of text, string functions on macro variables are handled implicitly:
 - Assignment: No quotes are necessary around the value of a macro variable (`%let mac_var = Hello;`)
 - Concatenation: `survey_&state` concatenates `&state` with "survey_"
- Most of the time, this is very convenient, but any time you avoid giving explicit instructions, computers may do something other than what you want!

Concatenation (cont'd)

Suppose we wished to import data from a file called "survey_IA.xls"

```
proc import datafile="H:\Data\survey_&state.xls"
out=survey_&state
replace;

run;
```

doesn't work, but

```
proc import datafile="H:\Data\survey_&state..xls"
out=survey_&state
replace;

run;
```

does

SAS Characters with Special Meaning

- Suppose we wish to assign a macro variable a string with semicolons, commas, or quotes
- The macro function `%str` can be used, for example, to pass an entire statement into a macro:

```
%macro reg(predictors, options);
proc reg data=dataset;
model outcome = &predictors;
&options;

run;
%mend reg;

%reg(age sex, %str(atest age, age - sex / canprint));
```

Evaluating Numeric Strings

- Remember, macro variables are strings, not numeric quantities:

```
%let sum = 1+1;
%put &sum;

1+1
```

- The function %eval can be used to obtain the (integer) numeric value of an expression containing macro variables:

```
%let total = %eval(&sum);
%put &total;

2
```

- Note: Floating point evaluations can be performed with %sysevalf

Getting More Out of Macros

Program Control

- The most powerful feature of macros is their ability to use conditional and iterative statements
- Data steps provide these same statements, but their effect is limited to a single data step
- Program control through macros can extend across multiple data steps and procedures

Conditional Statements

- Conditional statements in macros work just like those in data steps

```
%if (&state eq IA) %then %put Iowa;
%else %put Not Iowa;
```

%do Blocks

- Just as in data steps, compound statements are grouped using %do and %end:

```
%if (&state eq IA) %then
%do;
  %put Iowa;
  %put Corn grows here;
%end;
%else %put Not Iowa;
```

Iterative Statements

- Iterative macro statements will also be familiar to anyone who has used the data step versions:

```
%do i = 1 %to 10;
  %put %eval(&i**2);
%end;
```

- Note: %do...%while and %do...%until statements are also available

Macro Program Control Statements

- Macro program control statements are not valid in open code
- They must be contained within macros

Macro “Arrays”

- Suppose we created a list of states:

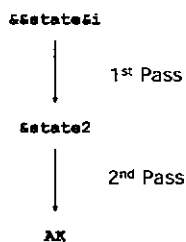
```
\let state1 = AL;
\let state2 = AK;
.
.
\let state50 = WY;
```

- If we were in the i^{th} iteration of a loop, how would we access the i^{th} member of the list?

```
\put &state&i;
IA2
```

Macro “Arrays” (cont'd)

- Instead, we must force the macro processor to make *multiple passes* over our code:



Example

- Suppose we wish to create a report by state of county rankings for a number of categories:

```
\macro report;
  \do i = 1 \to 50;
  \do j = 1 \to 25;
  \county_sort(&&var&j,
               state=&&state&i,
               order=descending);
  \end;
\end;
\mreport;
\report;
```

Nesting Macro Calls

- As we just saw, it is often a good idea to *nest* macro calls:

```
\macro a;
  SAS code...
\macro b;
  SAS code...
\mend a;
```

- It is not a good idea to nest macro definitions:

```
\macro a;
  SAS code...
  \macro b;
    SAS code...
  \mend b;
  SAS code...
\mend a;
```

Nesting Macro Calls (cont'd)

- When nesting macro calls, be careful to avoid variable collisions:

```
\macro print_sums;
  \do i = 1 \to 10;
  \put \sum(&i);
\end;

\macro sum(n);
  \let current_sum=0;
  \do i = 1 \to &val(&n);
  \let current_sum=current_sum +&i;
\end;
&val(&current_sum)
\mend;
```

- Scoping issues can be avoided by using %local to define macro variables

Interfacing With Data

- Suppose we submitted the following code to SAS:

```
data newdata;
  set survey_IA;
  %let AgeSq = Age**2;
run;
```

- What would happen?

How symput Works

- Calling the symput routine pauses execution of the data step and writes a data value to a macro variable
- Syntax:
`CALL SYMPUT('macro-variable', data-variable);`
- Both arguments to symput can be expressions
- IMPORTANT: You CANNOT access a macro variable within the same data step it is created

Example

- Suppose we want to compare two groups, but the preferred method depends on sample size:

```
%macro compare(dsn, class, cutoff=20);
data _null_;
  set &dsn nobs=nobs;
  call symputx('nobs',nobs);
  stop;
run;
if (&nobs < &cutoff) %then %do;
  proc npar1way data=&dsn;
    class &class;
  run;
%end;
%else %do;
  proc ttest data=&dsn;
    class &class;
  run;
%end;
%mend compare;

%compare(mydata, age);
```

Interfacing With Data (cont'd)

- Answer:

```
%put &AgeSq;
Age**2
```

- Because macros are resolved prior to the execution of a data step, special routines are required for macros to communicate with data:
 - symput puts data into a macro
 - symget extracts data from a macro

symputx: A Better symput

- CALL SYMPUTX is a variant of SYMPUT introduced in SAS 9 that has similar syntax, but handles the input of numeric values better
- The following example illustrates the difference between the two commands:

```
data _null_;
  call symput('symput',5);
  call symputx('symputx',5);
run;

%put |&symput|;
%put |&symputx|;

| 5|
|5|
```

How symget works

- symget is much more straightforward:
`data-variable = symget('macro-variable')`

Putting it all Together

- As a final example, suppose we want to create a list of indicator variables for the values of a categorical variable in a data set
- Note that if we don't know the values in advance, we have to approach the problem in two steps
 1. Determine the new variables we are to create
 2. Create a data set in which we assign values to the new variables

Putting it all Together (cont'd)

- We could approach the problem as follows:

```
%macro make_ind(dsn,cat);
proc sort data=dsn out=sorted;
  by cat;
run;
data _null_;
  set sorted end=eof;
  by cat;
  if first.cat then
    do;
      tot+1;
      call symputx("&cat.ind"||compress(tot),compress(&cat));
    end;
  if eof then call symputx('tot',tot);
run;
```

(cont'd)...

Putting it all Together (cont'd)

(cont'd)...

```
data dsn_ind;
  set dsn;
  %do i=1 %to %val(&tot);
    if (compress(&cat) eq "&cat.ind&i") then &cat.ind&i = 1;
    else &cat.ind&i = 0;
  %end;
run;
%mend make_ind;
```

Putting it all Together (cont'd)

```
%make_ind(survey_1A,city);
proc print data=survey_1A_ind;
run;
```

Obs	County	City	SBP	Age	Ames	Cedar Rapids	New Albin
1	Story	Ames	150	60	1	0	0
2	Linn	Cedar Rapids	180	45	0	1	0
3	Allamakee	New Albin	110	25	0	0	1
4	Story	Ames	120	50	1	0	0

References

- The SAS Macro Language Reference:
 - http://support.sas.com/documentation/onlinedoc/91pdf/index_912.html
- Carpenter, Art. 2004. *Carpenter's Complete Guide to the SAS® Macro Language, Second Edition*. Cary, NC: SAS Institute Inc.