Looping at a Higher Level – The %for Macro

Jim Anderson, Philip R. Lee Institute for Health Policy Studies, UCSF

ABSTRACT

There is a new looping feature in SAS®. It iterates over datasets, value lists, number ranges and macro arrays using the same simple syntax. It can repeat SAS steps in open code, reducing the need for single-call macros. It can generate large and small chunks of code: multiple steps, statements within a step and phrases within a statement. It unifies macro code and non-macro code making it easy to create data-driven programs[1]. This new SAS looping feature is the %for macro.

This paper illustrates a range of SAS coding problems that are solved using the %for macro. Examples of looping over the four kinds of data sources supported are given. The macro source code is included.

INTRODUCTION

The %for macro simplifies data-driven SAS programming. It repeatedly executes SAS code while looping through datasets and other data sources. It uses a common syntax for iterating over different kinds of data.

SAS has unique features for data-driven programming: call execute, the fetch function and single-call macros are three of many approaches to letting the data drive the processing. While these language features are effective, they are low-level implementation techniques unrelated to what a program is really doing. The *program plumbing* gets in the way of seeing the *program architecture*. The %for macro hides the plumbing.

THE %FOR MACRO

The %for macro is an iterative statement for repeatedly executing SAS code, both macro code and non-macro code. The general form of a %for macro call is as follows:

```
%for( <list of macro variables to be set for each observation>, <data source>,
     <SAS code to perform for each observation>
)
```

General form of %for macro call

The %for macro reads the data source sequentially. For each data source observation, the macro variables listed in the first parameter are assigned values from the observation and the SAS code is performed. Macro variable substitution is done in the SAS code each time the code is executed.

Four kinds of data sources can be used: (1) a SAS dataset, (2) a list of values, (3) a number range and (4) a list of macro arrays. Each kind of data source is identified by a different keyword parameter: “data=”, “values=”, “to=” and “array=”. Only one data source is permitted per %for call.

LOOPING OVER A NUMBER RANGE

A number range is a familiar data source – it’s used in a %do-%to-%by statement. Here is an example of a iterating over a range:

```
%for(i, to=5, do=%nrstr(
       proc sort data=dset&i; by key; run;
))
```

Example of a %for call with the “to=” data source

The code above sorts each of the 5 datasets named dset1 through dset5 by their “key” variable. The macro variable &i ranges over the values 1, 2, 3, 4 and 5. The “to=5” parameter specifies the upper limit in the range. When “from=” and “by=” keywords are omitted (as in this case) their default values are 1. The “do=” keyword parameter defines the SAS code to be repeated.

Notice that the code is contained in a %nrstr call. This is how the %for macro is able to get correct macro variable substitution in loop code. The %nrstr call prevents macro variable substitution in the “do=” code before the %for

macro is called, so that when each loop iteration is performed current values of the macro variables can be substituted in the code.

Here is a %for call with a range data source with “from=” and “by=” keywords specified.

```
%for(i, from=2, to=10, by=3, do=%nrstr(
       proc sort data=dset&i; by key; run;
))
```

Example of a %for call with the “to=” data source, including “from=” and “by=” parameters

The code above sorts datasets dset2, dset5 and dset8 by their “key” variable.

Why use the %for macro with the “to=” parameter when the %do %to statement gets the same results? The %for macro can be used in open code, eliminating the need to create a single-call macro.

LOOPING OVER A LIST OF VALUES

Lists are common in SAS programming. There are lists of variable names, numbers, datasets, etc. The %for macro iterates over value lists using the “values=” keyword. For example:

```
%for(table, values=Admission Process Outcome, do=%nrstr(
      proc import file="&hosp_xls_path" out=work.&table replace;
          sheet=”&table”;
      run;
  ))
```

Example of %for call with “values=” data source which imports 3 sheets from a spreadsheet

In the example above, proc import is called 3 times. Each of the 3 values in the “values=” list (Admission, Process and Outcome) is assigned to the macro variable “table”, and proc import is performed to import a sheet from a spreadsheet and create a dataset with the same name as the sheet.

When a %for call with a “values=” data source specifies multiple variables in the variable list, successive values from the value list are assigned to successive variables in the variable list before each loop iteration. For example:

```
%let histo_range= /* note: each line used in a separate iteration */
    IQI08  0  0.45  0.015
    IQI09  0  0.5   0.025
    IQI11  0  1     0.03
    IQI14  0  0.07  0.002
;

%for(measure endlo endhi endby, values=&histo_range, do=%nrstr(
    title "&measure Risk Adjusted Rate For All State Hospitals";
    proc univariate data psi_iqi(where=(measure="&measure")) noprint;
        histogram ra_rate / endpoints = &endlo to &endhi by &endby;
    run;
))
```

Example of %for call with “values=” data source which sets more than one macro variable

In the example above, the loop is executed four times, producing four histogram plots. The macro variables &measure, &endlo, &endhi and &endby are assigned successive values from the value list and the loop code is performed.

In case there are insufficient value list entries at the end of the list to assign a value to every variable list entry (i.e., the size of the value list is not a multiple of the size of the variable list), the “do=” code is not performed for the partially assigned list of variables.

LOOPING OVER SAS DATASETS

Datasets are the primary data source in SAS programming. The %for macro specifies a dataset data source with the "data=" parameter. The dataset may be qualified with "where" and "rename" clauses. For a dataset data source, the macro variable list must contain only dataset variable names. For each iteration of the %for loop, for each name in the macro variable list, the macro variable with that name is set to the value of the identically named dataset variable for that observation. For example:

```
%for(hospid hospname, data=report_hosps, do=%nrstr(
    title "&hospid &hospname Patient Safety Indicators";
    proc print data=psi_iqi(where=(hospid="&hospid")); run;
))
```

Example of %for call with "data=" data source and two macro variables

The example above loops over the dataset "report_hosps", which contains variables "hospid" and "hospname". For each observation in the dataset, macro variables &hospid and &hospname are assigned values from the corresponding dataset variables, and a report is printed.

A %for call with dataset data source can be used to generate statements within a SAS step. For example:

```
proc contents data=ready out=meta_ready noprint; run;

data up_ready;
    set ready;
    %for(name type length, data=meta_ready, do=%nrstr(
        %if &type=2 and &length<3 %then
        %do;
            &name = upcase(&name);
        %end;
    ))
run;
```

Example of %for call with "data=" data source - a loop that adds statements to a data step

The example above starts with a proc contents to create a dataset (meta_ready) which describes the variables in another dataset (ready). Then %for is called within the data step to add a convert-to-uppercase statement for those character variables with length less than 3. This illustrates using metadata (the output of proc contents) to selectively generate SAS code. It also shows that conditional macro statements can be used within the %for loop code.

LOOPING OVER MACRO ARRAYS

Macro arrays are a SAS programming convention that mirrors the utility of data step arrays[1, 2]. A macro array has a name and is represented by a collection of macro variables. Each array element is a macro variable whose name is the macro array name followed by the element's index number. For example, macro array ABC, with three elements ABC1, ABC2 and ABC3. The length of an array is contained in macro variable whose name is the macro array name followed by the letter N. For the example above, macro variable ABCN has value 3. Here is SAS code which creates macro array "meas":

```
*create macro array of measures;
%let meas1=PSI06;
%let meas2=PSI09;
%let meas3=PSI11;
%let meas4=PSI14;
%let measN=4;
```

Example of macro array creation with %let statements

(For a much simpler approach in creating macro arrays, see the %array macro [2].)

One or more macro arrays can be used as a %for macro data source with the “array=” keyword. When the %for variable list is present in a %for call, the number of variables must be the same as the number of macro arrays. The number of iterations to perform is defined by the length of the first macro array (or the “length=” parameter, if present). When multiple macro arrays are supplied, all macro arrays must be at least as long as the first array (or the “length=” parameter if present). Here is a %for example using data source macro array “meas” created above.

```
*combine reference datasets into single dataset;
data allrefs;
    set %for(psi, array=meas, do=%nrstr( refs_&psi(in=in_&psi) ));
    %for(psi, array=meas, do=%nrstr(
        if in_&psi then measure = "&psi";
    ))
run;
```

Example of %for call with “arrays=” data source, one array and one macro variable

The example above calls %for twice. The first call creates the list of datasets for the “set” statement. The second call sets variable “measure” to indicate the dataset source of each observation in the “allrefs” dataset. Omitting the macro variable list with an “array=” data source is a shortcut for the macro variable list being identical to the “array=” list. For example:

```
*combine reference datasets into single dataset;
data allrefs;
    set %for(array=meas, do=%nrstr( refs_&meas(in=in_&meas) ));
    %for(array=meas, do=%nrstr(
        if in_&meas then measure = "&meas";
    ))
run;
```

Example of %for call with “array=” data source, one array and an empty macro variable list

In the example above, macro variable &meas is assigned successive values from macro array “meas”.

NESTED LOOPS

Calls to the %for macro can be nested. For example:

```
*create one zip file per hospital system;
%for(system zipfilepath, data=systems, do=%nrstr(
    x ″del ″″&zipfilepath″″ ″;
    %for(hospreport, data=hospitals(where=(system=″&system″)), do=%nrstr(
        x ″pkzipc –add ″″&zipfilepath″″ ″″&hospreport″″ ″;
    ))
))
```

Example of nested %for calls – creating multiple zip files, each containing multiple report files

In the example above, the outer %for loop erases the old zip file (if present) and performs the inner loop, for each system. The inner %for loop adds one of more report files to the zip file for the current system. The end result is to produce multiple zip files, each containing multiple report files.

CONCLUSIONS

An advanced SAS technique (data-driven code) is made simple using the %for macro. The code is easy to read because the form is the same regardless of whether the data source is a dataset, a list of values, a number range or a group of arrays. The implementation details of iterating over different types of data sources are hidden, making the actual program logic more apparent. Finally, using a %for macro call for open-code looping is more concise and easier to read than embedding a %do loop in a single-call macro.

REFERENCES

[1] Art Carpenter (2004), *Carpenter's Complete Guide to the SAS Macro Language*, 2nd Edition, SAS Institute, Inc., Cary, NC.

[2] Ted Clay (2006), *Tight Looping with Macro Arrays*, Proceedings of the Thirty First Annual SAS Users Group International Conference, San Francisco, CA.

ACKNOWLEDGEMENTS

Ted Clay, Clay Software & Statistics, for showing me the feasibility of having SAS code as an argument to a macro call, and for the %array and %do_over macros.

CONTACT INFORMATION

Jim Anderson
Philip R. Lee Institute for Health Policy Studies
University of California, San Francisco
3333 California Street Suite 265
San Francisco CA 94118
E-mail: james.anderson@ucsf.edu.
Web: http://www.ucsf.edu

APPENDIX – MACRO IMPLEMENTATION

```
%macro for(macro_var_list,data=,values=,array=,to=,from=1,by=1,do=,delim=%str(
),length=);

/*

  Function: This macro performs a loop executing SAS code.  It
    proceeds sequentially through one of 4 possible data sources:
    (1) the records of a SAS dataset, (2) the values of a value
    list, (3) a range of integer values or (4) the elements of
    macro arrays. Data source values are assigned to macro
    variables specified in macro_var_list.

    Example:

      %for(hospid hospname, data=report_hosps, do=%nrstr(
          title ”&hospid &hospname Patient Safety Indicators”;
          proc print data=psi_iqi(where=(hospid=”&hospid”)); run;
      ))

  The example above loops over the dataset ”report_hosps”, which
  has dataset variables ”hospid” and ”hospname”.  For each dataset
  observation, macro variables &hospid and &hospname are assigned
  values from the identically named dataset variables and the loop
  code is performed, which prints a report.

  Parameters:
    macro_var_list = space-separated list of macro variable names
              to be assigned values upon each loop iteration.
    data     = Dataset name, with optional libname and where clause.
    values   = Delimiter-separated list of values (default delimiter
              is space)
    array    = Space-separated list of macro array names
    to       = Upper limit of iteration range.  The triple "to=",
              "from=" and "by=" defines the iteration range, as in
              a %do-%to-%by statement.  The presence of "to="
```

```
            specifies a range data source.
    from    = Lower limit of iteration range (default=1).
    by      = Increment in iteration range (default=1)
    do      = The SAS code to be generated for each iteration of
              the %for macro.  If macro variable substitution is
              to be done in the loop code (the typical case) enclose
              the code in a %nrstr() macro call to defer macro
              substitution to loop execution time.
    delim   = Delimiters for separating values (default is space).
    length  = Number of macro array elements to iterate over.
              When length= is omitted, the length is defined by the
              first macro variable name in the "array=" list.
              That first name with an "n" appended must be a macro
              variable containing the array length.

    Only 1 of the keyword parameters data=, values=, array=, or
    to= are permitted per %for call.  For each of these, the macro
    variable &_n_ is set to 1 for the first iteration, and incremented
    by one for each successive iteration.

    For data= iterations:
    For each observation in the dataset, all macro variables in the
    macro_var_list are assigned values of identically named dataset
    variables and the "do=" SAS code is executed.  The macro closes
    the dataset when end-of-dataset is reached.

    For values= iterations:
    The variables named in macro_var_list are assigned successive values
    from the values list. When all variables are assigned, the "do=" SAS
    code is executed.  The process repeats until the end of value list is
    reached.  If the end of value list is reached before all variables in
    macro_var_list are assigned values, then the iteration is terminated
    without performing the "do=" code on the partially assigned variables
    (the number of values in the value list should be a multiple of the
    number of names in macro_var_list).

    For array= iterations:
    The names in the macro_var_list are macro variables that will be
    assigned entries from the corresponding arrays in the "array="
    list for each loop iteration.

    For to= iterations:
    The first variable in the macro_var_list (if any) is assigned values
    as it would be by a %do-%to-%by statement.

    NOTE: Macro variable values obtained from datasets and arrays remain
    unquoted when they contain only letters, digits, whitespace, '_'
    and '.', otherwise they are quoted.  Values obtained from value list
    are always unquoted.

  Author: Jim Anderson, UCSF, james.anderson@ucsf.edu
    "Please keep, use and pass on the %more and %close_more macros
     with this authorship note.  -Thanks "

  Please send improvements, fixes or comments to Jim Anderson.

*/

  %global _for_loop_gen;
  %if &_for_loop_gen=%str( ) %then %let _for_loop_gen=0;
  %local _for_loop_itid _for_loop_ct _for_loop_code _for_loop_i
         _for_loop_val1 _for_loop_var1 _n_ _for_loop_set
         _for_loop_arrays _for_loop_values _for_loop_to
```

```
        _for_loop_var_num;
  %let _for_loop_set=%length(&data);
  %let _for_loop_arrays=%length(&array);
  %let _for_loop_values=%length(&values);
  %let _for_loop_to=%length(&to);
  %if (&_for_loop_set>0)+(&_for_loop_arrays>0)+
      (&_for_loop_values>0)+(&_for_loop_to>0)>1 %then
  %do;
    %put ERROR: "for" macro only one of "data=", "to=", "values=" or "array=" allowed;
    %return;
  %end;
  %let _for_loop_code=&do;
  %if %eval(%index(&do,%nrstr(%if))+%index(&do,%nrstr(%do))) %then
  %do; %* conditional macro code - need to embed in macro;
    %let _for_loop_gen=%eval(&_for_loop_gen+1);
    %unquote(%nrstr(%macro) _for_loop_&_for_loop_gen(); &do %nrstr(%mend;))
    %let _for_loop_code=%nrstr(%_for_loop_)&_for_loop_gen();
  %end;
  %let _for_loop_ct=0;
  %if &_for_loop_set %then
  %do; %* loop over dataset;
        %let _for_loop_itid=%sysfunc(open(&data));
        %if &_for_loop_itid=0 %then
        %do;
          %put ERROR: cant open dataset data=&data;
          %return;
        %end;
    %do %while(%sysfunc(fetch(&_for_loop_itid,NOSET))>=0);
      %let _for_loop_ct=%eval(&_for_loop_ct+1);
      %let _n_=&_for_loop_ct;
       %let _for_loop_i=1;
       %let _for_loop_var1=%scan(&macro_var_list,1,%str( ));
       %do %while(%str(&_for_loop_var1) ne %str( ));
         %let _for_loop_var_num=%sysfunc(varnum(&_for_loop_itid,&_for_loop_var1));
         %if &_for_loop_var_num=0 %then
         %do;
           %put ERROR: "&_for_loop_var1" is not a dataset variable;
           %return;
         %end;
         %if %sysfunc(vartype(&_for_loop_itid,&_for_loop_var_num))=C %then
         %do; %* character variable;
           %let _for_loop_val1=%qsysfunc(getvarc(&_for_loop_itid,&_for_loop_var_num));
         %if %sysfunc(prxmatch("[^\w\s.]+",&_for_loop_val1)) %then
           %let &_for_loop_var1=%qtrim(&_for_loop_val1);
         %else
           %let &_for_loop_var1=%trim(&_for_loop_val1);
         %end;
         %else
         %do; %* numeric variable;
           %let &_for_loop_var1=%sysfunc(getvarn(&_for_loop_itid,&_for_loop_var_num));
         %end;
         %let _for_loop_i=%eval(&_for_loop_i+1);
         %let _for_loop_var1=%scan(&macro_var_list,&_for_loop_i,%str( ));
       %end;
%unquote(&_for_loop_code)
    %end;
        %let _for_loop_i=%sysfunc(close(&_for_loop_itid));
    %return;
  %end;
  %else %if &_for_loop_arrays %then
  %do; %* loop over one or more arrays;
    %local _for_loop_arrays _for_loop_array1 _for_loop_len;
    %if &macro_var_list=%str( ) %then %let macro_var_list=&array;
```

```
    %let _for_loop_arrays=&array;
    %let _for_loop_array1=%scan(&array,1,%str( ));
    %if &length ne %str( ) %then
      %let _for_loop_len=&length;
    %else
    %do; %* getnumber of iterations from first macro array;
      %if %symexist(&_for_loop_array1.n) %then
        %let _for_loop_len=&&&_for_loop_array1.n;
      %else
      %do;
        %put ERROR: "for" macro for arrays needs "length=" argument;
        %return;
      %end;
    %end;
    %do _for_loop_ct=1 %to &_for_loop_len;
      %let _n_=&_for_loop_ct;
       %let _for_loop_i=1;
       %let _for_loop_var1=%scan(&macro_var_list,1,%str( ));
       %do %while(%str(&_for_loop_var1) ne %str( ));
         %let _for_loop_array1=%scan(&_for_loop_arrays,&_for_loop_i,%str( ));
         %if &_for_loop_array1=%str( ) %then
         %do; %* more variables than arrays;
              %put ERROR: "for" macro has more variables than arrays;
          %return;
         %end;
         %let _for_loop_val1=%superq(&_for_loop_array1&_n_);
         %if %sysfunc(prxmatch("[^\w\s.]+",&_for_loop_val1)) %then
           %let &_for_loop_var1=%qtrim(&_for_loop_val1);
         %else
           %let &_for_loop_var1=%trim(&_for_loop_val1);
         %let _for_loop_i=%eval(&_for_loop_i+1);
         %let _for_loop_var1=%scan(&macro_var_list,&_for_loop_i,%str( ));
       %end;
%unquote(&_for_loop_code)
    %end;
    %return;
  %end;
  %else %if &_for_loop_values ne 0 %then
  %do; %* loop over list of values;
    %local _for_value_index _for_loop_values _for_loop_delim _for_loop_extra;
    %let _for_loop_values=&values;
    %let _for_loop_delim=&delim;
    %let _for_value_index=1;
    %if &macro_var_list=%str( ) %then
    %do; %*empty variable list - perhaps (s)he just wants &_n_;
      %let macro_var_list=_for_loop_extra;
    %end;
    %do %while(1);
      %let _for_loop_ct=%eval(&_for_loop_ct+1);
      %let _n_=&_for_loop_ct;
       %let _for_loop_i=1;
       %let _for_loop_var1=%scan(&macro_var_list,1,%str( ));
       %do %while(%str(&_for_loop_var1) ne %str( ));
         %let
_for_loop_val1=%scan(&_for_loop_values,&_for_value_index,&_for_loop_delim);
         %let _for_value_index=%eval(&_for_value_index+1);
         %if %length(&_for_loop_val1)=0 %then
         %do; %* end of values before end of variables, terminate iteration;
          %return;
         %end;
         %let &_for_loop_var1=&_for_loop_val1;
         %let _for_loop_i=%eval(&_for_loop_i+1);
         %let _for_loop_var1=%scan(&macro_var_list,&_for_loop_i,%str( ));
```

```
      %end;
%unquote(&_for_loop_code)
    %end;
  %end;
  %else %if &_for_loop_to %then
  %do; %* loop from &from to &to by &by;
    %*local &macro_var_list;
    %let _for_loop_var1=%scan(&macro_var_list,1,%str( ));
    %let _for_loop_ct=1;
    %do _for_loop_i=&from %to %eval(&to) %by &by;
      %let _n_=&_for_loop_ct;
       %if %str(&_for_loop_var1) ne %str( ) %then %let &_for_loop_var1=&_for_loop_i;
       %let _for_loop_ct=%eval(&_for_loop_ct+1);
%unquote(&_for_loop_code)
    %end;
    %return;
  %end;
  %put ERROR: for macro requires a "data", "values", "to" or "array" keyword;
%mend for;
```