

# SAS Macros

## 1) Introduction

- SAS comes with a macro processor and a macro language that can make your programs more flexible
- macros provides a way of writing higher level statements and programs that generate SAS code
- in processing code
  - the macro processor reads the macro language and substitutes revised code into the SAS program
  - after this the generated SAS statements are executed
- two delimiters tell the macro processor that macro code is about to follow
  - *&name* – is used to indicate a macro variable
  - *%name* – is (generally) used to indicate a macro or a macro command<sup>1</sup>

## 2) Basics of SAS macro processing

- replacing text strings using macro variables
  - suppose that we wanted to write generalized code that annotated program output to indicate whether we had conducted a weighted or unweighted statistical analysis
  - we could use a macro %LET statement as follows

```
%LET wgtrun=Weighted Analysis;
```

- this would assign the text string “Weighted Analysis” to the macro variable *wgtrun*
- later in the program, we could use that macro variable as follows

```
TITLE "Descriptive Statistics - &wgtrun";
```

- the macro processor would take the text string “Weighted Analysis” and substitute it for *&wgtrun*, so that SAS would execute the statement

```
TITLE "Descriptive Statistics - Weighted Analysis";
```

- macro variables can be “global” or “local”
  - ◇ global variables are used throughout a SAS session
  - ◇ local variables are used within a particular macro
- generating SAS code using macros
  - there are macro commands, like the %LET statement described above
  - you can also create macros; the basic syntax is

```
%MACRO <macro_name>;
```

---

<sup>1</sup> The %INCLUDE, %LIST, and %RUN SAS commands follow this format but are not macros. These commands are more helpful in interactive line mode and will not be discussed here.

*macro definition here*  
%MEND <macro\_name>;

- the macro would substitute the text from the macro definition anywhere that the macro is invoked
- to invoke the macro you would type

```
%macro_name
```

- for example, typing

```
%MACRO wgt_type;  
    Weighted Analysis  
%MEND wgt_type;
```

```
TITLE "Descriptive Statistics - %wgt_type";
```

- would define a macro *wgt\_type* that would write the string, “Weighted Analysis,” wherever it was invoked
  - ◇ this would exactly replicate the macro string example that we examined earlier
  - ◇ generally we would use macros for something more sophisticated than this
- the macro statements can include SAS statements
- it is also possible to pass arguments into macros

```
%MACRO <macro_name>(<argument1=, ...argumentn=>>;  
    macro definition that includes &argument1, &argument2, ...  
%MEND <macro_name>;
```

- the arguments *argument1* through *argumentn* would be passed into the macro and used as local macro variables
- to invoke a macro with arguments, you would type

```
%macro_name(argument1=<value1>, ... argumentn=<valuen>);
```

- arguments make macros much more generalizable
- other macro features and techniques
  - macro statements can be used to generate SAS code conditionally

```
%IF <condition1> %THEN  
    %DO;  
        <conditional text 1>  
    %END;  
%ELSE <%IF <condition2> %THEN>  
    %DO;  
        <conditional text 2>  
    %END;
```

- the syntax is similar to a regular IF THEN ELSE statement but would be used within a macro definition
- repeated SAS code can also be generated

```
%DO <index_var> = <start_number> %TO <end_number>;
    <macro text to be repeated>
%END
```

- the macro language also has %DO %WHILE and %DO %UNTIL constructs

### 3) Macro variables

- system-defined variables
  - the macro processor has a number of variables already defined
  - examples,
    - ◇ &SYSDATE               contains the current date (with two digits for years)
    - ◇ &SYSDSN               contains the name of the most recently used SAS data set
  - the complete list is available at  
<http://support.sas.com/documentation/cdl/en/mcrolref/59526/HTML/default/a001071968.htm>
- user-defined macro variables
  - macro variables can be created in macro assignment statements using %LET
  - they can also be defined using %GLOBAL and %LOCAL statements
  - they can be created as arguments in macro definitions
  - there are some other statements (such as the iterative %DO statement) that create macro variables
- assigning values
  - in assignment (%LET) statements, the macro processor generally fills macro variables with the character string that follows, ignoring any leading or trailing blanks
  - for example, all three of these expressions are equivalent

```
%LET wgtrun=Weighted Analysis;
%LET wgtrun=  Weighted Analysis  ;
%LET wgtrun=
                    Weighted Analysis;
```

- the statements treat also treat most other characters, including digits, as text; for example,

```
%LET n=2+2;
```

puts the text string '2+2' into *n*

- the only exception to this are the characters % and &, which tell the system that a macro follows; for example,

```
%LET var1=value1;
```

```
%LET var2=&var1;
```

puts 'value1' into *var1* and subsequently *var2*

- assignment statements can also include macros and special macro functions
  - ◊ to perform integer evaluations of numbers, you can use the %EVAL() function

```
%LET n=%EVAL(2+2);
```

assigns the text string '4' to *n*

- ◊ to perform floating point (real) evaluations of numbers, use the %SYSEVALF() function
- a null value is assign by leaving the expression on the right side of the %LET statement blank,

```
%LET n=;
```

thereafter, any time that *&n* is used, it will generate nothing

- sometimes you want macro variables to contain special characters such as blanks, semicolons, parentheses, quotations, &, and %; this process is called macro quoting
  - ◊ the %STR() function masks all special characters except & and %
  - ◊ the %NRSTR() function masks these as well as & and %
- using macro variables
  - once a variable is created, you would use it (or resolve it) by typing it with an ampersand
  - for example

```
%LET wgtrun=Weighted Analysis;
```

```
TITLE "Descriptive Statistics - &wgtrun";
```

resolves to

```
TITLE "Descriptive Statistics - Weighted Analysis";
```

- an exception to this is a macro variable that is used within single quotes (this is an example where single and double quotes matter); for example

```
TITLE 'Descriptive Statistics - &wgtrun';
```

resolves to

```
TITLE 'Descriptive Statistics - &wgtrun';
```

the macro variable is effectively ignored

- macro variables can be used with other text

- ◇ macro variables can be appended to the end of a text expression; no delimiter is used

```
TITLE "Descriptive Statistics - Un&wgtrun";
```

resolves to

```
TITLE "Descriptive Statistics - UnWeighted Analysis";
```

- ◇ to append a macro variable to the beginning of a text expression, you would use a period as a delimiter

```
TITLE "Descriptive Statistics - &wgtrun.XX";
```

resolves to

```
TITLE "Descriptive Statistics - Weighted AnalysisXX";
```

- ◇ to append a macro variable to the beginning of a text expression and also include a period, you would use two periods as delimiters (this is useful for writing macros that reference external SAS data set); for example,

```
%LET libref=cartdata;
```

```
DATA &libref..hw1;
```

resolves to

```
DATA cartdata.hw1;
```

- listing the values of macro variables
  - you can write the values in macro variables to the SAS log using the %PUT statement
  - this is helpful in checking and debugging your code
- scope of macro variables
  - as mentioned, macro variables can be “global” (available throughout a program) or “local” (available only within a particular program)
  - generally macro variables that are created outside of a macro will automatically be global, while macro variables created inside of a macro or created as arguments for a macro will automatically be local
  - to specify that macro variables created inside of macro are global, you would use the %GLOBAL statement
  - if you try to reference a local variable outside the macro in which it is created, you will get a cryptic warning message that the “apparent symbolic reference” for that variable is “not resolved”
  - the macro variables that are currently available to the system are listed in the local and global Symbol Tables

- to write these variables you would use the %PUT statement with these options
  - ◇ `_ALL_` would write all of the macro variables
  - ◇ `_AUTOMATIC_` would write all of the automatically generated variables
  - ◇ `_GLOBAL_` would write all of the macro variables from the global table
  - ◇ `_LOCAL_` would write all of the macro variables from the local table
  - ◇ `_USER_` would write all of the user-created macro variables
- Passing information from a DATA step to a macro variable
  - sometimes it is useful to take values that are produced by the processing of the program and to pass these into subsequent macros
  - the SYMPUT routine does this; the syntax is

```
CALL SYMPUT(<macro_variable>, <value>);
```

- ◇ the *macro\_variable* argument is either a character string or a character variable with the name of the macro variable that you want to pass information into
- ◇ the *value* argument is a character string or character expression with the information that will be passed
- example,

```
DATA t1;
  SET t0 END=final;

  SAS commands

  IF final THEN
    CALL SYMPUT('numobs',TRIM(LEFT(_N_)));

  RUN;

  FOOTNOTE "Data set t0 has &numobs observations."
```

- passing information this way can be tricky because it raises issues of when information is available (note the use of the RUN statement before the FOOTNOTE statement above)

#### 4) Some debugging tools

- macros are trickier to program than regular SAS code
- there are some OPTIONS that help trace code
  - MLOGIC – causes SAS to trace the flow of the execution of macros
  - MPRINT – causes SAS to write each statement generated by a macro to the SAS log
  - SYMBOLGEN – causes SAS to describe how each macro variable resolves
- you can also use the %PUT statement to selectively write information to the SAS log
- note the trace options can generate a lot of output, so use them selectively and turn them off once you are sure that your macros are working correctly

- 5) For more information on macros, see <http://support.sas.com/documentation/cdl/en/mcrolref/59526/PDF/default/mcrolref.pdf>