

The George Washington University

Department of Statistics

STAT 4197/6197 – Fall 2019

Week 9 – October 25, 2019

Major Topic: SAS Macro Language Basics

Detailed Topics:

- 1) Macro Facility - Macro Variables vs. Macro Programs
- 2) Creating, Displaying , Referencing, and Deleting Macro Variables
- 3) Defining, Compiling, and Calling Macro Programs
- 4) Using Positional and Keyword Parameters in Macros
- 5) Converting Macro Variables into Data Step Variables

Acknowledgements: Portions of SAS' copyrighted SAS course content are reproduced here with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor. Parts of additional materials are adapted from SAS Documentation, and web-based resources. Please let me know of any errors or typos you have found. Do not copy or circulate the handouts

Readings:

1. Relevant Chapters/Sections - Delwiche L, and Slaughter S. *The Little SAS Book: A Primer*, Fifth Edition Paperback – November 7, 2012
2. Exercises from Relevant Chapters/Sections - Ottesen RA, Delwiche LD, and Slaughter SJ. *Exercises and Projects for The Little SAS Book*, Fifth Edition Paperback – July 1, 2015

Acknowledgements: Portions of SAS' copyrighted SAS course content are reproduced here with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor. Parts of additional materials are adapted from SAS Documentation, and web-based resources. Please let me know of any errors or typos you have found. Do not copy or circulate the handouts

3.

SAS Macro Facility Overview

The SAS macro facility enables you to use the SAS macro language to do the following:

- create macro variables that contain text, and reference them anywhere in a SAS program.



- write special programs (macros) that generate customized SAS code.



48

More specifically, the **macro facility** does the following:

- **Symbolic substitution within the SAS code**
- **Automated production of SAS code**
- **Conditional construction of SAS code**
- **Dynamic generation of SAS code**

SAS Macro Facility = Macro Processor + Macro Language

Macro Processor: The Part of Base SAS that performs the macro activity.

Macro Language: The syntax that interacts with the macro processor

Levels of Learning of Macro Language

- Code Substitutions
- Macro Statement and Functions
- Dynamic Programming

(See Carpenter, A. 2016. Carpenter's Complete Guide to the SAS Macro Language. Third Edition)

Acknowledgements: Portions of SAS' copyrighted SAS course content are reproduced here with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor. Parts of additional materials are adapted from SAS Documentation, and web-based resources. Please let me know of any errors or typos you have found. Do not copy or circulate the handouts

Words in SAS Language

There are four basic types of words or tokens

Name	Literal (i.e., characters enclosed in quotation marks)	Number	Special characters
<ul style="list-style-type: none"> DATA _var FIRSTOBS year_99 descending _n_ Percent8.2 	<ul style="list-style-type: none"> 'Stat6197' "1990-91" 'Mary Delany' "Final Exam" 	<ul style="list-style-type: none"> 3283 8.05 0b0x -5 6.4E-1 '04July18'd 	<ul style="list-style-type: none"> = ; ' + @ /

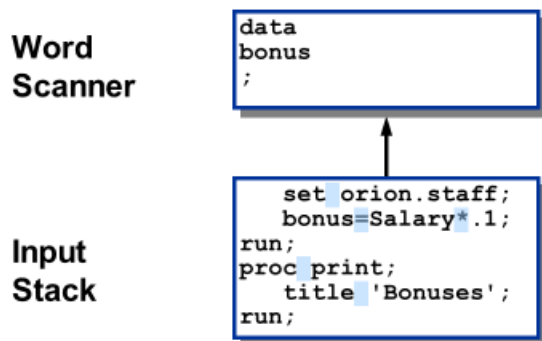
Point to remember: If you have both macro language and SAS language statements in the same step, the macro processor will execute macro language statements and then generate SAS code before any SAS language statements are executed.



Tokenization

A token ends when the word scanner detects

- a blank
- the beginning of another token.




The maximum length of a token is 32,767 characters.

Macro Triggers

During word scanning, two token sequences are recognized as *macro triggers*:

<code>&name-token</code>	a macro variable reference
<code>%name-token</code>	a macro statement, function, or call

-  The word scanner passes macro triggers to the *macro processor*. The macro processor will compile and execute macro triggers.

Creating and Referencing Macro Variables

Taking advantage of macro variables requires two steps.

1. Create and assign a value to a macro variable using one of these methods:
 - %LET statement in SAS code
 - INTO clause in a PROC SQL query
 - CALL SYMPUTX routine in SAS code
2. Reference the macro variable in SAS code so that SAS can resolve the macro variable value.
 - Use *¯o-name*

50

1. The %LET statement is used to create a macro variable manually and assign a value to it.
 - In open code
 - Within a macro program
2. The INTO clause in PROC SQL can automate creating macro variables
3. The CALL SYMPUTX in a data step can also automate creating macro variables.

[SAS Documentation]

[Problem Note 51984: A %LET statement might generate unexpected results when used to create a macro variable name in open code](#)

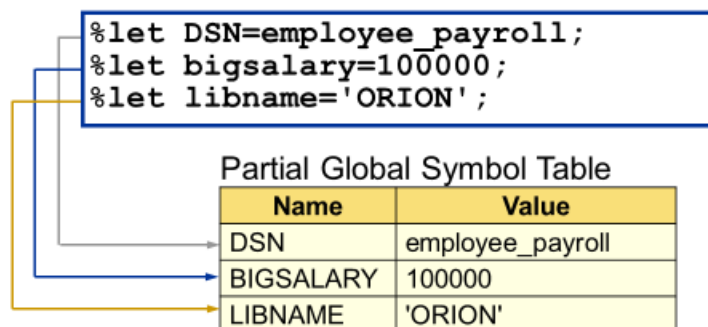
[%LET macro statement](#)

Acknowledgements: Portions of SAS' copyrighted SAS course content are reproduced here with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor. Parts of additional materials are adapted from SAS Documentation, and web-based resources. Please let me know of any errors or typos you have found. Do not copy or circulate the handouts

Creating Macro Variables: %LET Statement

If the macro variable does not exist, then the %LET creates it and assigns it a value.

If the macro variable exists (it is in the Global Symbol table), then the %LET assigns the new value.



52

Examples of the %LET Statement

Macro variable values have the following characteristics:

- Minimum length is 0 characters (null value).
- Maximum length is 65,534 characters (64K).
- Numbers are stored as text strings.
- Mathematical expressions are not evaluated.
- Case is preserved.
- Leading and trailing blanks are removed.
- Quotation marks, if any, are stored as part of the value.

```

%let year=2007;
%let city=Dallas, TX;
  
```

Name	Value
year	2007
city	Dallas, TX

18

User-Defined Macro Variables

User-defined macro variables have the following characteristics:

- Macro variable names follow SAS naming conventions.
- If the macro variable already exists, its value is overwritten.
- If the variable or value contain macro triggers, the triggers are evaluated before the %LET statement executes.

```
%let name=Ed Norton;
```

69

Macro Variable References

Macro variable references begin with an ampersand (&) followed by a macro variable name.

`&myvar` ← macro variable reference

Macro variable references

- are also called *symbolic references*
- can appear anywhere in a program
- are not case sensitive
- represent macro triggers
- are passed to the macro processor.

26

&mavar.

A macro variable can also be referenced with (1) an ampersand, the macro variable name, and a period.

Acknowledgements: Portions of SAS' copyrighted SAS course content are reproduced here with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor. Parts of additional materials are adapted from SAS Documentation, and web-based resources. Please let me know of any errors or typos you have found. Do not copy or circulate the handouts

Displaying Macro Variable Values

Use the %PUT statement to display the resolved macro variable value along with descriptive text in the SAS log.

```
%let DSN=employee_payroll;  
%let bigsalary=100000;  
%put DSN is &DSN;  
%put bigsalary is &bigsalary;
```

%PUT *text*;

Partial SAS Log

```
%put DSN is &DSN;  
DSN is employee_payroll  
%put bigsalary is &bigsalary;  
bigsalary is 100000
```

Using %PUT to Display Macro Variables

To display the names and values of all user-defined macro variables, use this form of the %PUT statement:

```
%put _user_;
```

Partial SAS Log

```
136 %put _user_;  
GLOBAL SITE Melbourne  
GLOBAL YEAR 2007
```

To display both user-defined and automatic macro variables, use this form of the %PUT statement:

```
%put _all_;
```

Partial SAS Log

```
136 %put _user_;  
GLOBAL SITE Melbourne  
GLOBAL YEAR 2007  
AUTOMATIC AFDSID 0  
AUTOMATIC AFDSNAME  
AUTOMATIC AFLIB
```

System-Defined Automatic Macro Variables

Automatic macro variables are set at SAS invocation and are always available. These include the following:

Name	Description
SYSDATE	Date of SAS invocation (DATE7.)
SYSDATE9	Date of SAS invocation (DATE9.)
SYSDAY	Day of the week of SAS invocation
SYSTIME	Time of SAS invocation
SYSSCP	Abbreviation for the operating system: OS, WIN, HP 64, and so on
SYSVER	Release of SAS software being used

To refer to a macro variable, use *¯o-variable-name*.

6

Macro Variables

SAS macro variables are stored in an area of memory referred to as the *global symbol table*.

There are two types of macro variables: automatic (created and updated by SAS) and user-defined.

Partial Global Symbol Table	
Name	Value
...	...
SYSLAST	NULL
SYSSCP	WIN
SYSTIME	09:00
SYSVER	9.3
DSN	employee_payroll

automatic →

user-defined →

The Use of the PUTLOG Statement vs. the %PUT Statement

```

1  *Ex3_putlog_PercentPut.sas;
2  options nodate nonumber nocenter
3      leftmargin=0.5in symbolgen;
4  %LET Path=C:\SASCourse\Week9;
5  LIBNAME perma "&Path";
6  data work.stocks;
7      set sashelp.stocks END=last;
8      count+1;
9      if last then putlog
10         @5 "Note: Number of observations=" count;
11  run;
12  /*old way to display the macro-variable-value */
13  %put Note: Macro Variable Path = &Path;
14
15  /*new way to display the macro-variable-value */
16  %put Note: Macro variable &=Path;

```

Line 3: The SYMBOLGEN option enables us to see that was substituted in the code echoed in the SAS log.

(Partial SAS Log)

```

      Note: Number of observations=699
NOTE: There were 699 observations read from the data set SASHELP.STOCKS.
NOTE: The data set WORK.STOCKS has 699 observations and 9 variables.
NOTE: DATA statement used (Total process time):
      real time           0.15 seconds
      cpu time            0.03 seconds

SYMBOLGEN: Macro variable PATH resolves to C:\SASCourse\Week9
12  /*old way to display the macro-variable-value */
13  %put Note: Macro Variable Path = &Path;
Note: Macro Variable Path = C:\SASCourse\Week9
14
15  /*new way to display the macro-variable-value */
16  %put Note: Macro variable &=Path;
SYMBOLGEN: Macro variable PATH resolves to C:\SASCourse\Week9
Note: Macro variable PATH=C:\SASCourse\Week9

```

Points to remember: The %PUT statement is valid in open code. It writes to column one of the next line in the SAS log. It writes a blank line if no text is specified.

Acknowledgements: Portions of SAS' copyrighted SAS course content are reproduced here with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor. Parts of additional materials are adapted from SAS Documentation, and web-based resources. Please let me know of any errors or typos you have found. Do not copy or circulate the handouts

PUTLOG (or PUT) vs. %PUT

PUTLOG or PUT can write the following to the SAS log

- text strings inside quotation marks
- values of the variables found in the DATA step

(Ex17_put_putlog.sas)

- Use the PUTLOG statement to write informational message (including the debugging message) to the SAS log.
- Use the PUT statement to write to an external file that is specified in the FILE statement.

The %PUT statement

- writes text strings and values of the macro variables to the SAS log, starting in column one
- writes a blank line if text is not specified
- does not require quotation marks around text
- is valid in open code
- can appear
 - before the DATA step
 - after the DATA step
 - in the middle of the DATA step

Substitution within a SAS Literal

Double quotation marks enable macro variable resolution, and single quotation marks prevent macro variable resolution.

```
12 %let site=Melbourne;
13 proc print data=orion.employee_addresses;
14   where City="&site";
15   var Employee_ID Employee_Name;
16   title 'Employees from &site';
17 run;
NOTE: There were 41 observations read from the data set
ORION.EMPLOYEE_ADDRESSES.
WHERE City='Melbourne';
```

Site resolved in double quotation marks.


Employees from &site		
Obs	Employee_ID	Employee_Name
2	120145	Aisbitt, Sandy
24	120168	Barcoe, Selina

Site did not resolve in single quotation marks.

Leading Text

Leading text is never a problem.

```
%let month=jan;
proc chart data=orion.y2010&month;
  hbar week / sumvar=sale;
run;
proc plot data=orion.y2010&month;
  plot sale*day;
run;
```




```
proc chart data=orion.y2010jan;
  hbar week / sumvar=sale;
run;
proc plot data=orion.y2010jan;
  plot sale*day;
run;
```

101

Adjacent Macro Variable References

Adjacent macro variable references are never a problem.

```
%let year=2010;
%let month=jan;
proc chart data=orion.y&year&month;
  hbar week / sumvar=sale;
run;
proc plot data=orion.y&year&month;
  plot sale*day;
run;
```



```
proc chart data=orion.y2010jan;
  hbar week / sumvar=sale;
run;
proc plot data=orion.y2010jan;
  plot sale*day;
run;
```

102

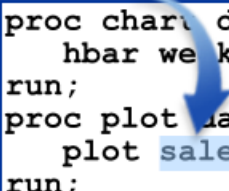
Trailing Text

Trailing text can be a problem. Why?

Why is trailing text **not** a problem here?

```
%let year=2010;
%let month=jan;
%let var=sale;
proc chart data=orion.y&year&month;
  hbar week / sumvar=&var;
run;
proc plot data=orion.y&year&month;
  plot &var*day;
run;

proc chart data=orion.y2010jan;
  hbar week / sumvar=sale;
run;
proc plot data=orion.y2010jan;
  plot sale*day;
run;
```



103

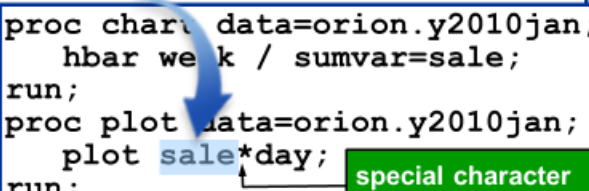
Trailing Text

Trailing text is a problem if it changes the reference.

It is not a problem here because of the special character.

```
%let year=2010;
%let month=jan;
%let var=sale;
proc chart data=orion.y&year&month;
  hbar week / sumvar=&var;
run;
proc plot data=orion.y&year&month;
  plot &var*day;
run;

proc chart data=orion.y2010jan;
  hbar week / sumvar=sale;
run;
proc plot data=orion.y2010jan;
  plot sale*day;
run;
```



104

Trailing text can be a problem if a dot is not added after the macro variable.

See `Ex6_Join_macro_var_text.sas` on GitHub.

Acknowledgements: Portions of SAS' copyrighted SAS course content are reproduced here with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor. Parts of additional materials are adapted from SAS Documentation, and web-based resources. Please let me know of any errors or typos you have found. Do not copy or circulate the handouts

Indirect References to Macro Variables

Because the **custID** value matches the numeric suffix of another macro variable, **custID** can indirectly reference the other macro variable.

Symbol Table	
Variable	Value
CUSTID	9
NAME4	James Kvarniq
NAME5	Sandrina Stephano
NAME9	Cornelia Krahel
⋮	⋮

49

Indirect References to Macro Variables

The Forward Rescan Rule

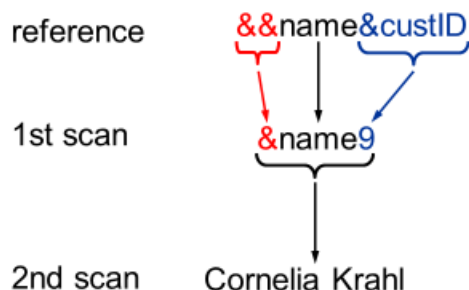
- Multiple ampersands preceding a name token denote an indirect reference.
- Two ampersands (&&) resolve to one ampersand (&).
- The macro processor rescans an indirect reference, left to right, from the point where multiple ampersands begin.
- Scanning continues until no more references can be resolved.

50

Acknowledgements: Portions of SAS' copyrighted SAS course content are reproduced here with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor. Parts of additional materials are adapted from SAS Documentation, and web-based resources. Please let me know of any errors or typos you have found. Do not copy or circulate the handouts

Indirect References to Macro Variables

The indirect reference causes a second scan.



52

Indirect References to Macro Variables

The **custID** macro variable indirectly references a **name** macro variable.

Symbol Table	
Variable	Value
CUSTID	9
NAME4	James Kvarniq
NAME5	Sandrina Stephano
NAME9	Cornelia Krah
⋮	⋮

Scan sequence:

`&&name&custID` → `&name9` → **Cornelia Krah**

53

Example: Referencing Macro Variables Indirectly Using a Macro

The double ampersands always resolve into a single ampersand.

```

1 *Ex7_indirect_reference_1.sas;
2 Options symbolgen;
3 %LET disease1=cvd;
4 %LET disease2=cancer;
5 %LET disease3=stroke;
6 %LET disease4=hbp;
7 %LET disease5=diabetes;
8 %LET last_element=5;
9 %macro ref;
10     %DO i = 1 %TO &last_element;
11         %put &&disease&i;
12     %END;
13 %mend ref;
14 %ref

```

Line 3-8: Six macro variables are created

Line 11: Note the two &s in front of the macro variable name

When the macro %REF is executed, SAS:

- (1st iteration) Resolves **&&disease&i** to **&disease1** in the first scan
&disease1 to **cvd** in the second scan.
- (2nd iteration) Resolves **&&disease&i** to **cancer**.
- ...
- (5th iteration) Resolves **&&disease&i** to **diabetes**.

Also see [Ex15_Indirect_ref.sas](#) on GitHub.

[Ex5_Lookup_mvar.sas](#) on GitHub.

Why do we use macro variables?

One reason for using macro variables is to circumvent the problem that the variable stored in a SAS data set cannot be retrieved in a DATA step without using it as an input data set.

Problem:

```

1  *Ex1_Motivation_for_macro_variables;
2  proc means data=sashelp.class mean maxdec=1;
3    var weight;
4    output out=stats mean=average_wgt;
5  run;
6
7  data test;
8    set SASHELP.class;
9    *This line of code does not work;
10   weight_ratio=weight/average_wgt;
11  run;

```

The variable average_wgt stored in data set STATS cannot be used in the DATA step (lines 7-11) above. See the solution in the same SAS program on GitHub.

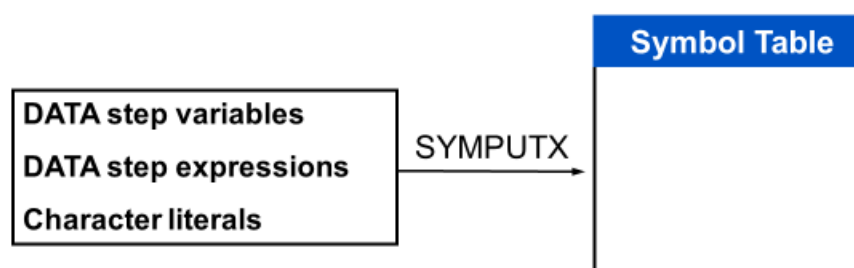
Creating Macro Variables in DATA Step: The SYMPUTX Routine

SYMPUTX Routine

The SYMPUTX routine assigns to a macro variable any value available to the DATA step during execution time.

It can create macro variables with the following:

- static values
- dynamic (data dependent) values
- dynamic (data dependent) names



Global Symbol Table

Global macro variables can be created by any of the following:

- %LET statement
- DATA step SYMPUTX routine
- PROC SQL SELECT statement INTO clause
- %GLOBAL statement

73

%GLOBAL Statement

General form of the %GLOBAL statement:

```
%GLOBAL macro-variable1 macro-variable2 . . . ;
```

- The %GLOBAL statement adds one or more macro variables to the global symbol table with null values.
- It has no effect on variables already in the global table.
- It can be used anywhere in a SAS program.

74

Local Symbol Table

Local macro variables can be

- created at macro invocation (parameters)
- created during macro execution
- updated during macro execution
- referenced anywhere within the macro.

76

Local Symbol Table

A *local symbol table* is

- created when a macro with a parameter list is called or a local macro variable is created during macro execution
- deleted when the macro finishes execution.

Macros that do not create local variables do not have a local table.

75

Ex14_symlocal_symglobal on GitHub

Acknowledgements: Portions of SAS' copyrighted SAS course content are reproduced here with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor. Parts of additional materials are adapted from SAS Documentation, and web-based resources. Please let me know of any errors or typos you have found. Do not copy or circulate the handouts

%LOCAL Statement

Declare the index variable of a macro loop as a local variable to prevent accidental contamination of a like-named macro variable in the global table or another local table.

```
%macro putloop;
  %local i;
  %do i=1 %to &numrows;
    %put Country&i is &&country&i;
  %end;
%mend putloop;
```

80

m105d13

Local Symbol Table

Local macro variables can be created by the following within a macro definition:

- %LET statement
- DATA step SYMPUTX routine
- PROC SQL SELECT statement INTO clause
- %LOCAL statement

78

Scope of the Macro Variables

Macro variables are global when they are defined outside of a macro; and local to a macro when they are defined inside of a macro when the %GLOBAL statement is not used.

```

1  *Ex8_Global_Local.sas;
2  *Code Adapted from Carpenter (2016);
3  option symbolgen;
4  %LET Year_outside = 2005;
5
6  %macro one;
7    %global Year_inside_one;
8    %LET Year_inside_one = 2006;
9    %PUT &Year_inside_one;
10 %mend one;
11
12 %macro two;
13   %LET Year_inside_two = 2007;
14   %PUT &Year_inside_two;
15 %mend two;
16
17 %macro last;
18   %one
19   %two
20   %put &Year_outside &Year_inside_one;
21   %put &Year_inside_one;
22 %mend last;
23 %last

```

Line 4: &Year_outside is a GLOBAL macro variable.

Line 8: &Year_inside_one is also a GLOBAL macro variable.

Line 14: &Year_inside is a LOCAL macro variable.

Line 20: The %PUT statement demonstrate these availabilities.

[19] ". /Ex8_Global_Local.sas"

Acknowledgements: Portions of SAS' copyrighted SAS course content are reproduced here with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor. Parts of additional materials are adapted from SAS Documentation, and web-based resources. Please let me know of any errors or typos you have found. Do not copy or circulate the handouts

Deleting Macro Variables Using %SYMDEL and SYMDEL Routine

```

1  *Ex21_Delete_Macro_Variables.sas;
2  options SYMBOLGEN nodate nonumber;
3  * Create macro variables;
4  %let mvar1=var_name1;
5  %let mvar2=var_value2;
6  %let mvar3=var_label;
7  %let mvar4=var_name2;
8  %let mvar5=var_value2;
9  %put _user_;
10 %symdel mvar3;
11 %put _user_;
12
13 %macro delete_m;
14     %if %symexist (mvar4) %then %symdel mvar4;
15     %if %symexist (mvar5) %then %symdel mvar5;
16 %mend delete_m;
17 %delete_m
18 %put _user_;
19
20 data _null_;
21     call symdel('mvar1', 'nowarn');
22     call symdel('mvar2', 'nowarn');
23 run;
24 %put _user_;

```

Line 10: The %SYMDEL removes user-defined macro variables from the global symbol table.

Line 14-15: The macro statements can delete user-defined macro variables.

Lines 21- 24: The SYMDEL routine can also be used to remove user-defined macro variables from the global symbol table.

What Is a Macro Definition?

Like macro variables, macros generate text. However, macros can contain programming logic to dynamically control what text is generated and when it is generated. Macros can also accept parameters.

A macro or macro definition can store the following:

- macro language statements or expressions
- complete or partial SAS statements
- complete or partial SAS steps
- any text
- any combination of the above




37

Defining a Macro

This code defines the **Time** macro, which displays the current time.

```
%macro time;
  %put The current time is %sysfunc
    (time(),timeampm.) .;
%mend time;
```

```
%MACRO macro-name;
  macro-text
%MEND <macro-name>;
```

-  Macro names follow SAS naming conventions and cannot be reserved names such as names of macro statements or functions (for example, LET and SCAN).

38

psm08d06

Ex12_GetInfo_Host_Computer.sas on GitHub

Ex13_Macro.sas on GitHub



Macro Compilation

To verify macro compilation, specify the MCOMPILENOTE=ALL option.

```
OPTIONS MCOMPILENOTE=ALL|NONE;
```

```
options mcompilenote=all;
%macro time;
  %put The current time is %sysfunc
    (time(),timeampm.);
%mend time;
```

SAS Log

```
1 options mcompilenote=all;
2 %macro time;
3   %put The current time is %sysfunc
4     (time(),timeampm.);
5 %mend time;
NOTE: The macro TIME completed compilation without errors.
      3 instructions 76 bytes.
```

8

The default value is MCOMPILENOTE=NONE. m103d01



Calling a Macro

To call a macro, precede the macro name with a percent sign (%).

```
%time
```

```
%macro-name
```

SAS Log

```
178 %time
The current time is 2:49:39 PM.
```

A macro call

- can appear anywhere (similar to a macro variable reference)
- represents a macro trigger
- is passed to the macro processor
- is **not** a statement (no semicolon required)
- causes the macro to execute.

9

m103d01

Acknowledgements: Portions of SAS' copyrighted SAS course content are reproduced here with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor. Parts of additional materials are adapted from SAS Documentation, and web-based resources. Please let me know of any errors or typos you have found. Do not copy or circulate the handouts

Macro with Positional and Keyword Parameters

```

10 *Ex16_posi_key_para_macro.sas;
11 %MACRO printdata(dsn, num=);
12     PROC PRINT DATA=&dsn (obs=&num) noobs;
13 RUN;
14 %MEND;
15 %printdata(SASHELP.CLASS, num=7)
16 %printdata(SASHELP.CARS, num=5)
17 %printdata(SASHELP.RETAIL, num=10)

```

Lines 2-5: Define the macro.

Line 3: Specify the keyword parameters in the %macro statement.

Lines 6-7: Invoke the same macro in three separate macro calls, supplying to each call parameter values.

The following is repeated from page 4.

Macro Compilation Process	Macro Execution Process
<ul style="list-style-type: none"> The user submits the macro definition (%MACRO to %END sandwich). The macro facility <ul style="list-style-type: none"> intercepts the macro definition performs a macro compilation The compiled macro definition gets stored in WORK.SAMACR 	<ul style="list-style-type: none"> SAS checks for the existence of the macro in the catalog WORK.SASMACR. If the macro exists, it gets executed. The macro call is replaced by any text generated by the macro Generated code is compiled and executed by Base SAS.

Source: Carpenter (2016).

Ex19_Macro_Vars_Resolve.sas on GitHub.

Ex18_Week_9_List_of_Files.sas on GitHub