The George Washington University
Department of Statistics

STAT 6197 – Spring 2019
Week 9 – March 22, 2019

Major Topic: Macro Facility and Macro Tools in SAS

Detailed Topics:

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

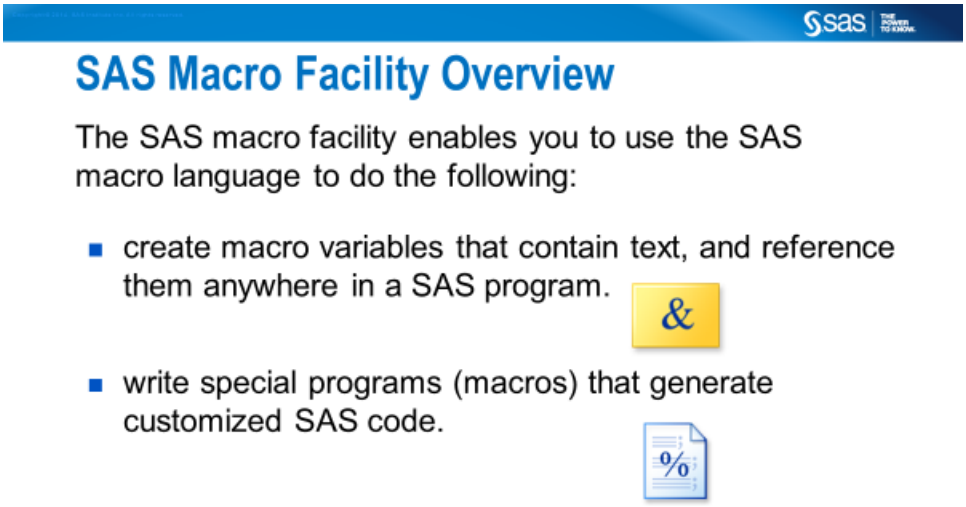
Readings:

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

.

3.



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

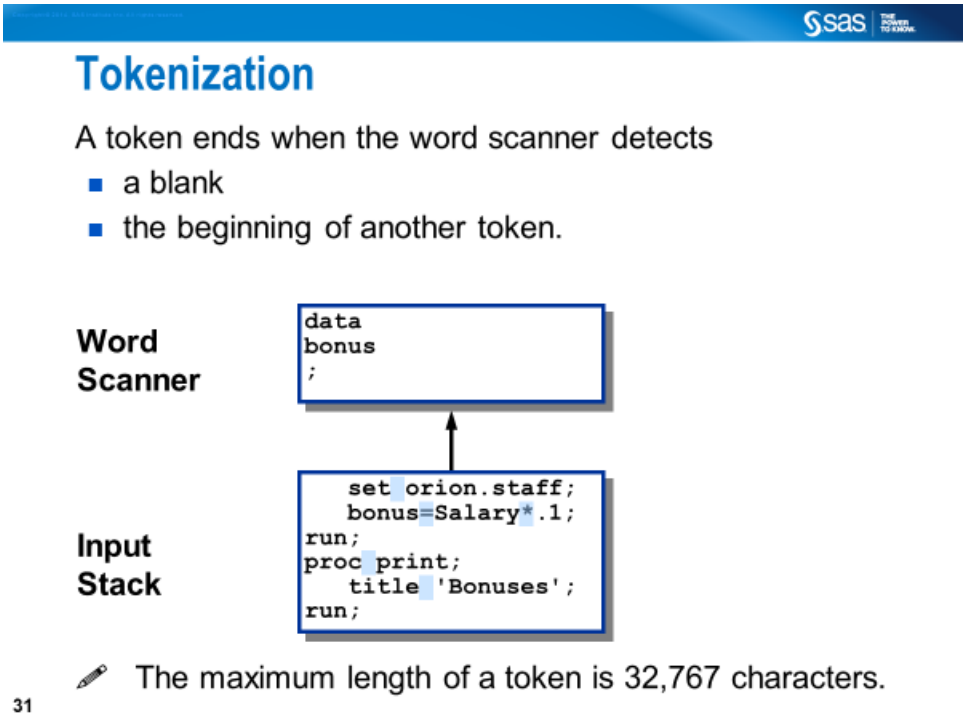
(See Carpenter, A. 2016. Carpenter's Complete Guide to the SAS Macro Language. Third Edition)

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
• DATA • _var • FIRSTOBS • year_99 • descending • _n_ • Percent8.2	• 'Stat6197' • "1990-91" • 'Mary Delany' • "Final Exam"	• 3283 • 8.05 • 0b0x • -5 • 6.4E-1 • '04July18'd	• = • ; • ' • + • @ • /

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.



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 Triggers

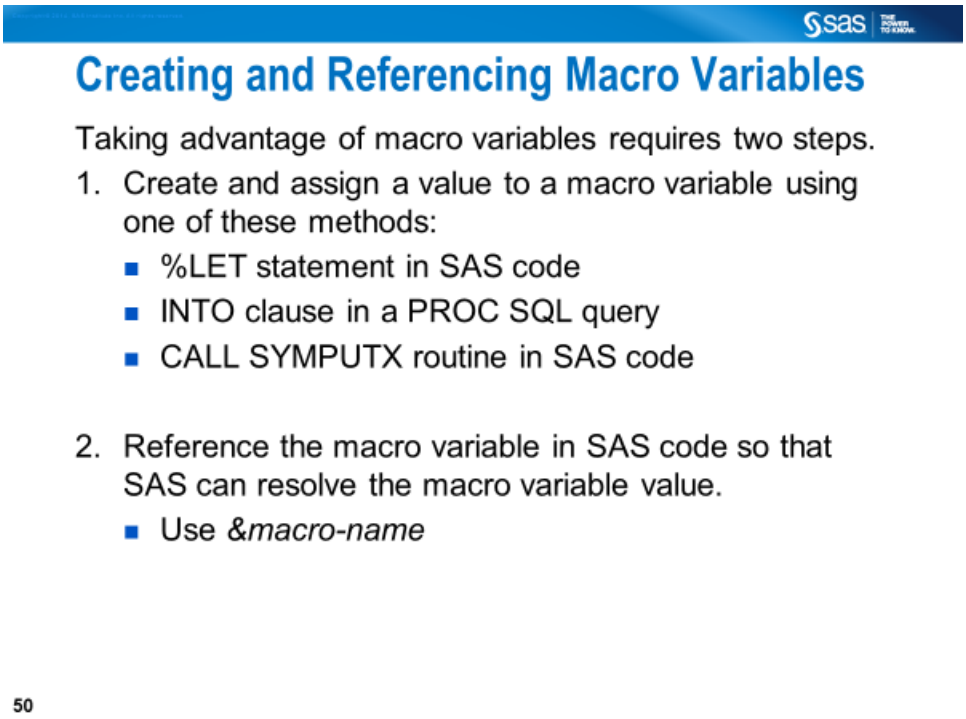
During word scanning, two token sequences are recognized as *macro triggers*:

&*name-token*	a macro variable reference
%*name-token*	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.

32

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



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

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.

```
%let DSN=employee_payroll;
%let bigsalary=100000;
%let libname='ORION';
```

Partial Global Symbol Table

Name	Value
DSN	employee_payroll
BIGSALARY	100000
LIBNAME	'ORION'

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.

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
```

64 s108d05

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

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
```

28

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

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 *&macro-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 → SYSLAST, SYSSCP, SYSTIME, SYSVER

user-defined → DSN

49

Additional Examples: %LET and %PUT Statements

```
1  *Ex2_percent_let.sas;
2  %LET CITY1=   Washington DC;
3  %LET CITY2=   "Washington DC";
4  %LET CITY3=   Washington   DC;
5  %LET m_sum=2+2;
6  %LET msum_eval=%eval(2+2);
7  %LET CALC_FRAC=%SYSEVALF(5.5/2);
8  %LET amount=%sysfunc(putn(5000000, dollar14.));
9  %put &=CITY1;
10 %put &=CITY2;
11 %put &=CITY3;
12 %put &=m_sum;
13 %put &=msum_eval;
14 %put &=CALC_FRAC;
15 %put &=amount;
```

	Line 3: Leading and trailing blanks from the value are removed. Line 2: Quotation marks are part of the macro variable value.

143 %put &=CITY1; CITY1=Washington DC 144 %put &=CITY2; CITY2="Washington DC" 145 %put &=CITY3; CITY3=Washington DC 146 %put &=m_sum; M_SUM=2+2 147 %put &=m_sum_eval; M_SUM_EVAL=4 148 %put &=CALC_FRAC; CALC_FRAC=2.75 149 %put &=amount; AMOUNT=$5,000,000	Line 4: Embedded extra blanks are not removed Line 5: Mathematical expressions in macro variable values do not get evaluated. Line 6: With the %EVAL function, mathematical expressions … do get evaluated. Line 7: Only with the %SYSEVALF function, expressions with fractional numbers … get evaluated. Line 8: %SYSFUNC is a function that executes SAS functions. PUTN assigns the value based on the value in the first argument and the format in the second argument.

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

The use of macro variables for text substitution within the SAS code

```
 1  *Ex4_Text_Substitution.sas;
 2  ***Part 1: No macro variables used;
 3  proc contents data=SASHELP.CLASS;
 4  run;
 5  title "Data Set: SASHELP.CLASS";
 6  proc print data=SASHELP.CLASS (obs = 5);
 7  run;
 8
 9  ***Part 2: Macro variables used;
10  options symbolgen;
11  %let dsn = SASHELP.CLASS;
12  %let HowMany = 5;
13  %put _user_;
14  proc contents data=&dsn;
15  run;
16  title "Data Set: &dsn";
17  proc print data=&dsn (obs = &HowMany);
18  run;
```

Lines 11-12: When the %LET statements is executed, the text values of the macro variable is stored in the global symbol table.

Lines 14, 16-17: The macro trigger is passed to the macro processor to search the symbol table for the reference. The macro processor resolves the macro variable references (&dsn and &HowMany) substituting their respective values (sashelp.class, and 5, respectively), passing the resolved value to the Input Stack.

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

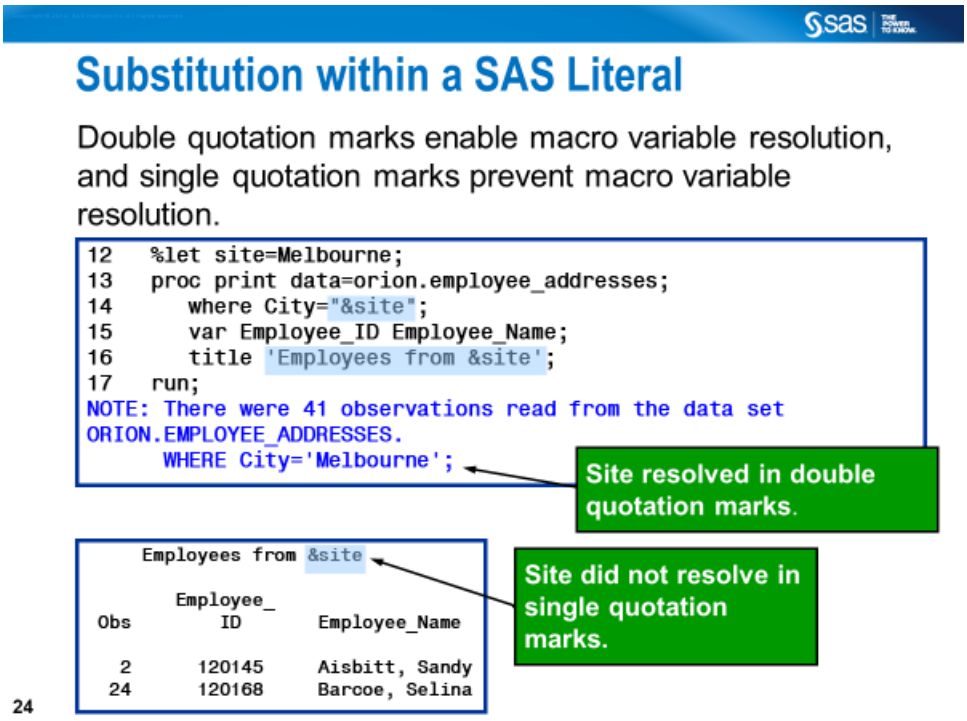
(Ex17_put_putlog.sas)

- Use the PUTLOG statement to write informational message (includding the debugging message) to the SAS log.

- Use the PUT statement to write to an external file that isspecified 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



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

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

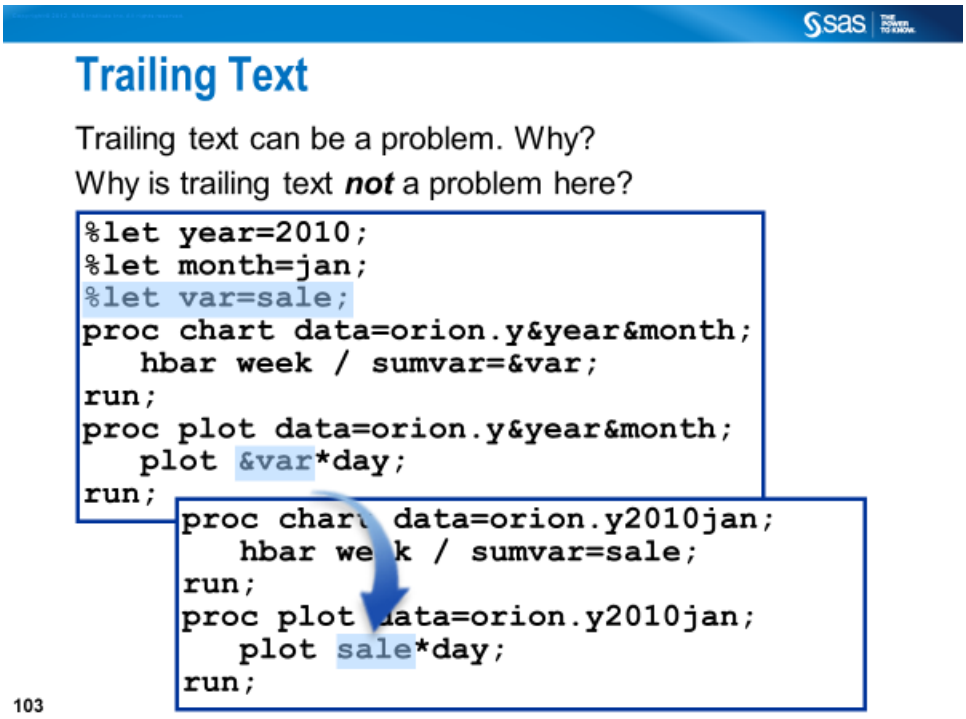
Adjacent macro variable references are never a problem.

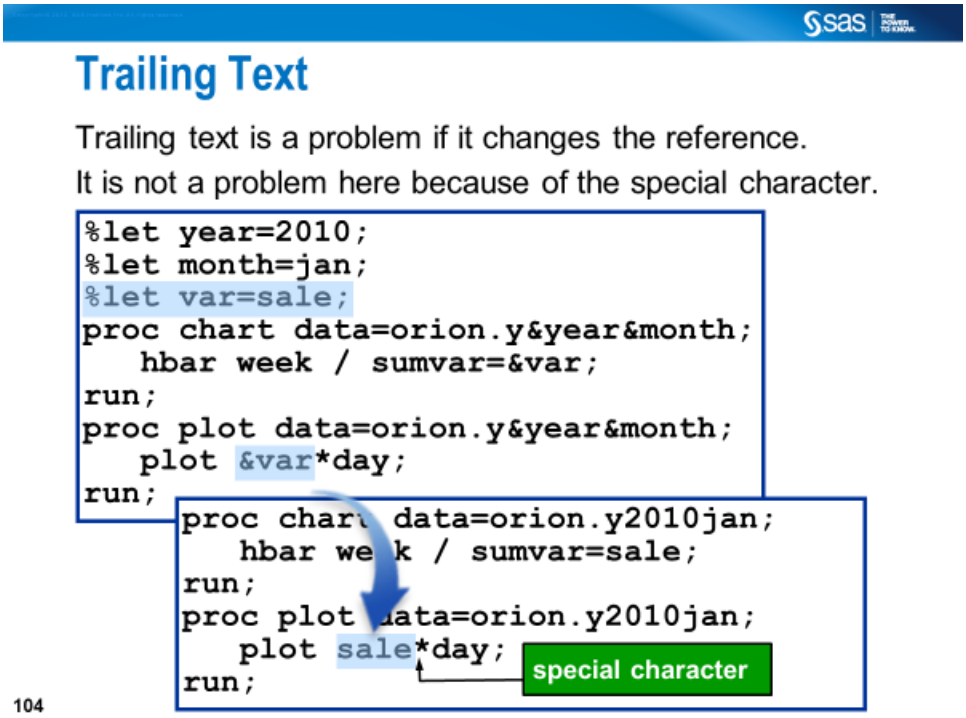
```
%let year=2010;
%let month=jan;
proc chart data=orion.y&year&month;
   hbar week / sumvar=sale;
run;
proc plot data=orion.y&year&month;
   plot sale*day;
run;
```

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

102

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





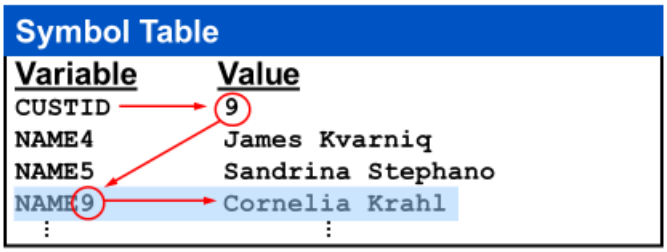
Trailing text can be a problem if a dot is not added after the macro variable.

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.



49

Indirect References to Macro Variables

The Forward Rescan Rule

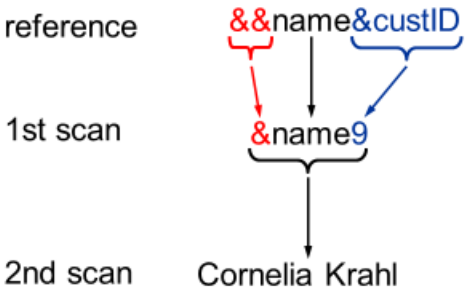
- Multiple ampersands preceding a name token denote an indirect reference.
- Two ampersands (&&) resolve to one ampersand (&).
- The macro processor rescans an indirect reference, left to right, from the point where multiple ampersands begin.
- Scanning continues until no more references can be resolved.

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 Krahl

Scan sequence:

&&name&custID → &name9 → Cornelia Krahl

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 **&&diseasei** to **cancer**.
  …
- (5th iteration) Resolves **&&disease&i** to **diabetes**.

```
SYMBOLGEN:  Macro variable LAST_ELEMENT resolves to 5
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable I resolves to 1
SYMBOLGEN:  Macro variable DISEASE1 resolves to cvd
cvd
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable I resolves to 2
SYMBOLGEN:  Macro variable DISEASE2 resolves to cancer
cancer
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable I resolves to 3
SYMBOLGEN:  Macro variable DISEASE3 resolves to stroke
stroke
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable I resolves to 4
SYMBOLGEN:  Macro variable DISEASE4 resolves to hbp
hbp
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable I resolves to 5
SYMBOLGEN:  Macro variable DISEASE5 resolves to diabetes
diabetes
```

Also see Ex15_Indirect_ref.sas on GitHub.

Ex5_Lookup_mvar.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

Why do we use macro variables?

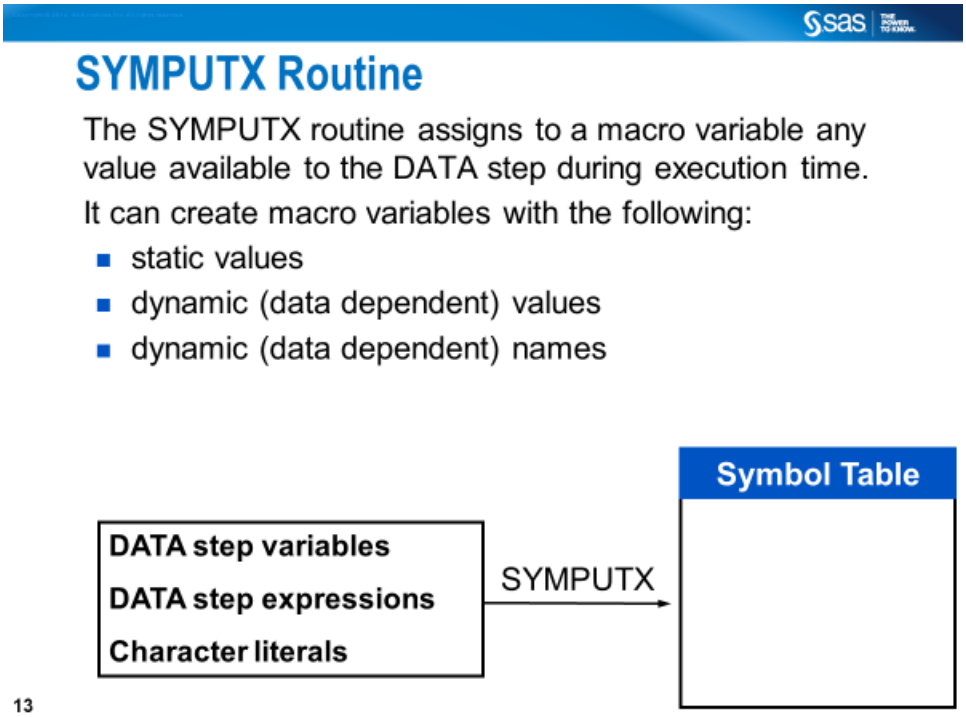
One reason for using macro variables is to circumvent the problem that the variable stored in a SAS data set cannot be retrieved in a DATA step without using it as an input data set.

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



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

```
1  *** Ex10_call_symputx_1.sas;
2  Options nodate nonumber;
3  proc means data=sashelp.class noprint;
4  var weight;
5  output out=mystats mean=mean_weight;
6  run;
7
8  data _null_;
9  set mystats;
10 call symputx('AverageWeight', mean_weight);
11 run;
12
13 %put &AverageWeight;
14 %put _GLOBAL_;
15
16 proc print data=sashelp.class noobs;
17 var name sex weight;
18 where weight > &AverageWeight;
19 title1 'Passing Values between Steps';
20 title2 'Average Body Weight:';
21 title3 "%sysfunc(putn(&AverageWeight, 5.1)) lbs";
22 run;
```

Passing Values between Steps
Average Body Weight:
100.0 lbs

Name	Sex	Weight
Alfred	M	112.5
Carol	F	102.5
Henry	M	102.5
Janet	F	112.5
Mary	F	112.0
Philip	M	150.0
Robert	M	128.0
Ronald	M	133.0
William	M	112.0

Line 10: This SYMPUTX routine has two parameters separated by a comma inside its parentheses. The first parameter, a constant enclosed in quote marks is the name of the macro variable being created. The second parameter is the mean value of WEIGHT (from SASHELP.CLASS) to the macro variable.

Lines 15 and 18: The macro variable used in the both WHERE and TITLE statements.

The SYMPUTX routine is a DATA step statement that works during DATA step execution time.

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

Symputx Routine with a Conditional Logic

```
82  *Ex10_create_macro_vars_call_symputx_1.sas;
83  data Have;
84    set SASHELP.CLASS end=last;
85     if age >14 then do;
86      n+1;
87     if last then call symputx('number', n);
88     output;
89    end;
90  run;
91  Footnote "There are &number of observations with AGE>14";
92  proc print data=have; run;
```

Line 87: During the last iteration of the DATA step when LAST=1, the SYMPUTX routine creates a macro variable named NUMBER whose value is the value of the DATA step variable N. The DATA step variable N is assigned to the macro variable NUMBER.

Line 91: The value of the macro variable &Number is inserted into the FOOTNOTE statement where AGE is greater than 14.

Ex10_call_symputx_1.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

The SYMPUTX Routine: Creating a Numbered Series of Macro Variables

```
94  *Ex10_create_macro_vars_call_symputx_1.sas;
95  DATA _NULL_;
96   set sashelp.class;
97   call symputx('Name' || STRIP(put( _N_, 2.)), Name);
98  run;
99  %put &Name1 &Name2 &Name3;
```

Line 97: This SYMPUTX routine has two parameters separated by a comma inside its outer parentheses. The first parameter, a constant enclosed in quote marks conctaenated with the value of the automatic variable _N_, will be the name of the macro variable being created. The second parameter will be the value of the DATA step vatribale NAME being assigned to each macro variable. The code creates 19 macro variables, and the value of of each of those macro variables is a character string. These macro variable reside in the _GLOBAL_ symbol table.

```
Alfred Alice Barbara
```

Creating a Single Macro Variable with Multiple Values Using PROC SQL INTO

```
10  *Ex11_one_multiple_mvars_sql.sas;
11  Options nosymbolgen;
12  proc sql noprint;
13   select distinct make
14          INTO :makes separated by ','
15          FROM SASHELP.CARS;
16  quit;
17  %put List of Values of Car Make (Unique) : &makes;
```

```
113  %put List of Values of Car Make (Unique) : &makes;
List of Values of Car Make (Unique) :
Acura,Audi,BMW,Buick,Cadillac,Chevrolet,Chrysler,Dodge,Ford,GMC,Honda,Hummer,Hyundai,Infiniti
,Isuzu,Jaguar,Jeep,Kia,Land
Rover,Lexus,Lincoln,MINI,Mazda,Mercedes-Benz,Mercury,Mitsubishi,Nissan,Oldsmobile,Pontiac,Por
sche,Saab,Saturn,Scion,Subaru,Suzuki,Toyota,Volkswagen,Volvo
```

See Ex11_create_macro_vars_sql.sas on GitHub for more examples.

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 a Series of Macro Variables Using PROC SQL

```
*Ex11_one_multiple_mvars_sql.sas;
%let Put_title = List of
Values into a Series of Macro
Variables;
proc sql noprint;
 select distinct make
        INTO :makes1-
         FROM SASHELP.CARS;
 %put Number of Rows: &sqlobs;
quit;
%macro reveal;
 %put &Put_title;
 %Do i=1 %TO &Sqlobs;
    %put &&makes&i;
  %end;
%mend reveal;
%reveal
```

(Partial log)

```
119   %put Number of Rows: &sqlobs;
Number of Rows: 38
```

```
List of Values into a Series of Macro
Variables
Acura
Audi
BMW
Buick
Cadillac
Chevrolet
Chrysler
Dodge
Ford
GMC
Honda
Hummer
Hyundai
Infiniti
Isuzu
Jaguar
Jeep
Kia
Land Rover
Lexus
Lincoln
MINI
Mazda
Mercedes-Benz
Mercury
Mitsubishi
Nissan
Oldsmobile
Pontiac
Porsche
Saab
Saturn
Scion
Subaru
Suzuki
Toyota
Volkswagen
```

Notice that the macro variables follow an INTO clause and are preceded by a colon. Here the PROC SQL creates a series of macro variables, each with its own distinct value (often referred to as *vertical list* of macro variables of the form &S2, and so on. When PROC SQL is executed, a series of macro variables including &SQLOBS is generated and placed in the most local symbol table. The value of the macro variable &SQLOBS is the number of rows processed by the SELECT statement.

When you submit this macro, it is compiled and then stored in the default catalog **work.sasmacr**.

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 a Single Macro Variable with One Value Using PROC SQL INTO

```
1  *Ex11_one_multiple_mvars_sql.sas;
2  proc sql noprint;
3    select count(*)
4           INTO :nobs
5           FROM SASHELP.CARS;
6  quit;
7  %put Number of Observations = %SYSFUNC(LEFT(&nobs));
8  run;

SYMBOLGEN:  Macro variable NOBS resolves to      428
81    %put Number of Observations = %SYSFUNC(LEFT(&nobs));
Number of Observations = 428
82    run;
```

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

Passing Information between SAS steps

The PROC SQL code below uses the SELECT and INTO statements to store the value of average weight [ave(weight)] in a macro variable. This macro variable is then grabbed into a subsequent data step to calculate the ratio of the individual weight to the overall average weight. The average weight is also displayed in the title.

```
 1 *Ex9_macro_vars_transfer.sas;
 2 options nodate nonumber ps=58;
 3 proc sql noprint;
 4  select avg(weight) format=6.2
 5     into :m_avg_wt
 6     from sashelp.class;
 7  quit;
 8 %put &m_avg_wt ;
 9 data class;
10   set sashelp.class;
11   ratio_wt=weight/&m_avg_wt;
12 run;
13 title "Class Data: Average Weight &m_avg_wt lbs";
14 proc print data=class noobs;
15  var name weight ratio_wt;
16  run;
```

Line 5: m_avg_wt is the macro variable.

Line 8: This statement on this line is used to display the macro variable in SAS log.

Line 11: The same macro variable, which resides in the GLOBAL symbol table (not in the data set, is used in the denominator in ratio_wt calculations.

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.

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

```
 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"
```

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.

Deleting global macro variables

https://blogs.sas.com/content/sastraining/2018/05/07/deleting-global-macro-variables/

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

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

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.
```

- The default value is MCOMPILENOTE=NONE.

8 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

Macro with a Single Positional Parameter

```
1  *Ex16_posi_key_para_macro.sas;
2  options obs=5;
3  %macro printit(dsn);
4    proc print data=&dsn;
5    run;
6  %put _local_;
7  %mend printit;
8  %printit (sashelp.class)
```

Line 3: The %macro statement declares the macro **printit** and defines its parameter **dsn** in parentheses. Note that the parameter dsn is a local macro variable.

Lines 4-5: Macro "Body" (proc step in this example).

Line 6: This statement is used to display the value of the local macro variables. Below is the partial SAS Log.

```
PRINTIT DSN sashelp.class
```

In the above SAS log,

- PRINTIT is the name of the symbol table.
- DSN is the name of the macro variable.
- Sashelp.class is the current value of the macro variable.

(Reference: Carpenter, 2016; p. 418)

Line 7: End of the macro definition marked by the %mend statement; specifying the macro name on this statement is a good programming style, but not required.

Line 8: There is no semicolon after the macro call. The parameter value – sashelp.class - is supplied when the macro is called.

Note that the macro variable DSN is assigned a value sasahelp.class and written to the local symbol table for %PRINTIT after the %**prinit (sashelp.class)** is executed.

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
• The user submits the macro definition (%MACRO to %END sandwich). • The macro facility - intercepts the macro definition - performs a macro compilation • The compiled macro definition gets stored in WORK.SAMACR	• 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).

Converting Macro Variables into Data Step Variables
(Adapted from Carpenter, 2016)

The SYMGET function is used to convert the current value of a macro variable to the character value a data step variable within a DATA step, with a default length of 200 (Carpenter, 2016).

```
 1 *Ex20_SYMGET_RESOLVE.sas;
 2 data Have;
 3 Quiz1_score=70;
 4 call symputx('weight', .05);
 5 wgt1=symget('weight');
 6 W_Quiz1_score=Quiz1_score*symget('weight');
 7 W_Quiz1_score_x=Quiz1_score*input(symget('weight'), best12.);
 8 run;
 9 proc contents data=Have;run;
10 proc print data=Have;run;
```

(Partial SAS Log)

```
NOTE: Character values have been converted to numeric values at the places given by:
      (Line):(Column).
      16:27
NOTE: The data set WORK.HAVE has 1 observations and 4 variables.
```

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
1	Quiz1_score	Num	8
3	W_Quiz1_score	Num	8
4	W_Quiz1_score_x	Num	8
2	wgt1	Char	200

Obs	Quiz1_ score	wgt1	W_Quiz1_ score	W_Quiz1_ score_x
1	70	0.05	3.5	3.5

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

SYMGET Function vs. RESOLVE Function

Note that the argument of the SYMGET function is the macro variable itself and that argument of the RESOLVE function is a macro variable reference.

```
12 *Ex20_SYMGET_RESOLVE.sas;
13 %let factor=.05;
14 data Have2;
15 var_factor1=&factor;
16 var_factor2 = symget('factor');
17 var_factor3= resolve('&factor');
18 run;
19 proc contents data=Have2;run;
```

(Partial Output)

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
1	var_factor1	Num	8
2	var_factor2	Char	200
3	var_factor3	Char	200

Ex19_Macro_Vars_Resolve.sas on GitHub.

Ex18_Week_9_List_of_Files.sas on GitHub