

Chapter 1 Overview of the Macro Language

1.1	Introduction to the Macro Facility	1-3
1.2	Introduction to Macro Variables	1-5
1.3	Introduction to Macros	1-12

1.1 Introduction to the Macro Facility

Purpose of the Macro Facility

The **macro facility** is a text processing facility for automating and customizing flexible SAS code.

The macro facility supports

- symbolic substitution within SAS code
- automated production of SAS code
- dynamic generation of SAS code
- conditional construction of SAS code.

3

Efficiency of Macro-Based Applications

The macro facility can reduce program

- development time
- maintenance time.

SAS code generated by macro techniques

- does not compile or execute faster than any other SAS code
- depends on the efficiency of the underlying SAS code, regardless of how the SAS code was generated.

4

Components of the Macro Facility

The macro facility enables you to

- create and resolve **macro variables** (*symbolic variables*) anywhere within a SAS program
- write and resolve **macro programs** (*macros*) that generate custom SAS code.

1.2 Introduction to Macro Variables

What is a Macro Variable?

Macro variables store text, including

- complete or partial SAS steps
- complete or partial SAS statements.

Macro variables are referred to as *symbolic variables* because SAS programs can reference macro variables as symbols for additional program text.

7

Macro Symbol Tables

Macro variables are stored in an area of memory called a *symbol table*.

When SAS is invoked, a global symbol table is created and initialized with **automatic macro variables**.

User-defined macro variables can be added using macro programming statements.

Global Symbol Table	
Automatic Variables	SYSDATE9 18MAY2007
	SYSTIME 13:05
	SYSDAY Friday
User-defined Variables	GENDER F
	DATE 14JAN1990
	AGE 12

8

Characteristics of Macro Variables

Macro variables

- have a minimum length of 0 characters (*null value*)
- have a maximum length of 65,534 (64K) characters
- store numeric tokens as character strings.

Macro variables stored in the global symbol table persist throughout a SAS program.

9

Macro Variable References

Macro variable references

- begin with an ampersand (&) followed by a macro variable name
- represent macro triggers
- are also called *symbolic* references
- can appear anywhere in your program
- are passed to the macro processor for resolution.

10

Macro Variable Resolution: Find the Name

When the macro processor receives a macro variable reference, it searches the symbol table for the macro variable name.

Your Submitted Code

```
proc print data=sashelp.class;
title1 'Listing of SASHELP.CLASS Data Set';
title3 "Created at &systemtime on &sysdate9";
run;
```

Macro Processor

Global Symbol Table

SYSDATE9	18MAY2007
SYSTEMTIME	13:05
SYSDAY	Friday

11

Macro Variable Resolution: Substitution

If the macro processor locates the matching macro variable name, the associated text value is substituted for the symbolic reference.

Code to be Executed

```
proc print data=sashelp.class;
title1 'Listing of SASHELP.CLASS Data Set';
title3 "Created at 13:05 on 18MAY2007";
run;
```

Macro Processor

Global Symbol Table

SYSDATE9	18MAY2007
SYSTEMTIME	13:05
SYSDAY	Friday

12

Failed Macro Variable Resolution

If the matching macro variable name cannot be located, the macro processor issues a warning to the SAS log. The original symbolic reference remains intact.

Your Submitted Code

```
proc print data=sashelp.class;
title1 'Listing of SASHELP.CLASS Data Set';
title3 "Created at &sysclock on &sysdate9";
run;
```

Code to be Executed

```
proc print data=sashelp.class;
title1 'Listing of SASHELP.CLASS Data Set';
title3 "Created at &sysclock on 18MAY2007";
run;
```

SAS Log

```
WARNING: Apparent symbolic reference SYSCLOCK not resolved.
```

13

Macro References within Double Quotes

If you need to resolve a macro variable within a literal, enclose the literal in **double** quotes.

Your Submitted Code

```
proc print data=sashelp.class;
title1 'Listing of SASHELP.CLASS Data Set';
title3 "Created at &systemtime on &sysdate9";
run;
```

Macro Processor

Code to be Executed

```
proc print data=sashelp.class;
title1 'Listing of SASHELP.CLASS Data Set';
title3 "Created at 13:05 on 18MAY2007";
run;
```

14

Macro References within Single Quotes

Macro variable references within single quotes are not passed to the macro processor. Therefore, they do not resolve.

Your Submitted Code

```
proc print data=sashelp.class;
title1 'Listing of SASHELP.CLASS Data Set';
title3 'Created at &systemtime on &sysdate9';
run;
```

Macro Processor

Code to be Executed

```
proc print data=sashelp.class;
title1 'Listing of SASHELP.CLASS Data Set';
title3 'Created at &systemtime on &sysdate9';
run;
```

15

Macro References Outside of a Literal

If you need to resolve a macro variable outside of a literal, no additional quotes are required.

Your Submitted Code

```
proc print data=work.&sysday;
title1 "Listing of work.&sysday Data Set";
title3 "Created at &systemtime on &sysdate9";
run;
```

Global Symbol Table

SYSDATE9	18MAY2007
SYSTIME	13:05
SYSDAY	Friday

Macro Processor

Code to be Executed

```
proc print data=work.Friday;
title1 "Listing of work.Friday Data Set";
title3 "Created at 13:05 on 18MAY2007";
run;
```

16

Create a Macro Variable

The %LET statement creates a user-defined macro variable and assigns it a value in the symbol table.

General form of the %LET statement:

```
%LET variable=value;
```

- *variable* follows SAS naming conventions.
- If *variable* already exists, its *value* is overwritten.
- If *variable* or *value* contain macro references, the references are resolved before the assignment is made.

17

The %LET Statement: Rules for Values

Value can be any string:

- maximum length is 65,534 (64K) characters
- minimum length is 0 characters (*null value*)
- numeric tokens are stored as character strings
- mathematical expressions are **not** evaluated
- the case of *value* is preserved
- quotes bounding literals are stored as part of *value*
- leading and trailing blanks **are removed** from *value* before the assignment is made.

18

Creating a User-Defined Macro Variable

The %LET statement is executed within the macro processor. No classic SAS code is generated.

Your Submitted Code `%let dsn=sashelp.class;`

Macro Processor

Global Symbol Table	
SYSDATE9	18MAY2007
SYSTIME	13:05
SYSDAY	Friday
DSN	sashelp.class

19

Referencing a User-Defined Macro Variable

You can reference a macro variable multiple times in a SAS program. The resolved text will be consistent.

Your Submitted Code

```
proc print data=&dsn;
title1 "Listing of &dsn Data Set";
title3 "Created at &systemtime on &sysdate9";
run;
```

Global Symbol Table	
SYSDATE9	18MAY2007
SYSTIME	13:05
DSN	sashelp.class

Macro Processor

Code to be Executed

```
proc print data=sashelp.class;
title1 "Listing of sashelp.class Data Set";
title3 "Created at 13:05 on 18MAY2007";
run;
```

20

1.3 Introduction to Macros

What is a Macro Definition?

Macro definitions (or *macros*) enable you to implement more complex macro applications.

Macro definitions store text, including

- complete or partial SAS steps
- complete or partial SAS statements
- macro programming statements.

22

Defining a Macro

General form of a macro definition:

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

The %MACRO statement begins a macro definition and names the macro.

The %MEND statement ends a macro definition.

23

Defining a Macro

General form of a macro definition:

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

macro-name follows SAS naming conventions.

macro-text can include any text, including

- SAS statements or steps
- macro variables, functions, statements, or calls
- any combination of the above.

24

Macro Compilation: Behind the Scenes

When a macro definition is submitted,

- macro language statements are
 - checked for syntax errors
 - compiled
- SAS statements, macro variables and other text are
 - **not** checked for syntax errors
 - stored exactly as typed
- the compiled macro is stored as an entry in a SAS catalog, **work.sasmacr** by default.
- The macro is **not** executed.

25

Compiling a Macro

Example: Submit a macro definition to compile the macro.

```
%macro prtlast;  
  proc print data=&dsn;  
    title1 "Listing of &dsn";  
  run;  
%mend prtlast;
```

By default when a macro is successfully compiled, there are no log messages generated.

The `MCOMPILENOTE=` system option controls whether such messages are displayed.

26



The SAS statement `options mcompilenote=all;` will write notes to the SAS log upon completion of the compilation of any macro. The default is NO.

Macro Execution

Macro program executes when it is *called*.

Macro calls

- begin with a percent sign (%) followed by a macro name
- represent macro triggers
- are also considered symbolic references
- can appear anywhere in your program
- are passed to the macro processor for resolution
- are **not** SAS statements, that is, do not require a semicolon.

27

Calling a Macro

Example: Assign a value to the macro variable DSN and call the PRTLAST macro.

```
%let dsn=sashelp.class;
%prtlast
```

Note that the macro call does not end with a semicolon.

28

Resolution into Macro Definition Text

The macro call resolves to the text stored in the macro definition. There are several macro variable references stored as part of this macro definition.

Your
Submitted
Code

```
%let dsn=sashelp.class;
%prtlast
```

WORK.SASMACR.PRTLAST.MACRO

```
proc print data=&dsn;
title "Listing of &dsn";
run;
```

Macro Processor

Initial macro
resolution

```
proc print data=&dsn;
title "Listing of &dsn";
run;
```

29

Resolution of Embedded Macro Variables

The macro variables in the macro definition are resolved before the code it sent to the SAS compiler.

Initial macro resolution

```
proc print data=&dsn;
title "Listing of &dsn";
run;
```

Global Symbol Table

DSN	sashelp.class
-----	---------------

Macro Processor

Code to be executed

```
proc print data=sashelp.class;
title "Listing of sashelp.class";
run;
```

30

SAS Log for Macro Calls

The SAS log reflects that a PROC PRINT step executed after the macro was resolved. By default, the SAS code generated by the macro does not appear in the SAS log.

Partial SAS Log

```
%prtlast
```

```
NOTE: There were 19 observations read from the data set SASHELP.CLASS.
```

31

Displaying SAS Code Generated by a Macro

The MPRINT option writes to the SAS log any SAS code generated as a result of macro execution.

General form of the MPRINT|NOMPRINT option:

```
OPTIONS MPRINT;  
OPTIONS NOMPRINT;
```

The default setting is NOMPRINT.

32

Generated Code Displayed through MPRINT

Example: Activate the MPRINT option before calling the macro.

Partial SAS Log

```
options mprint;  
%prtlast  
MPRINT(PRTLAST):  proc print data=sashelp.class;  
MPRINT(PRTLAST):  title1 "Listing of sashelp.class";  
MPRINT(PRTLAST):  run;  
  
NOTE: There were 19 observations read from the data set SASHELP.CLASS.
```

33

Macro Parameters

To simplify coding, macros can be defined with a parameter list. There are two types of parameter lists

- positional parameters
- keyword parameters.

Macro parameter values are stored in a local symbol table that is independent of the global symbol table.

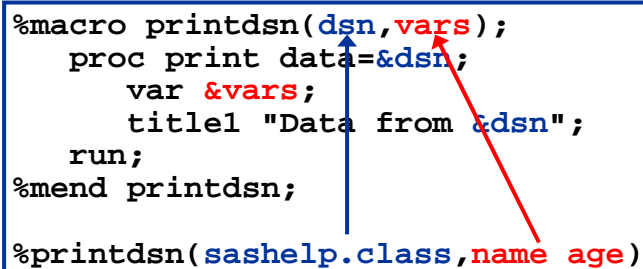
34

Positional Macro Parameters

Positional parameters use a one-to-one correspondence between

- parameter names supplied in the macro definition
- parameter values supplied in the macro call.

```
%macro printdsn(dsn,vars);  
  proc print data=&dsn;  
    var &vars;  
    title1 "Data from &dsn";  
  run;  
%mend printdsn;  
  
%printdsn(sashelp.class,name age)
```



35

Positional Parameters: Definition

General form of a **macro definition** with a positional parameter list:

```
%MACRO macro-name (parameter-1, ... , parameter-n);  
    macro text  
%MEND <macro-name>;
```

Positional parameter **names** are

- enclosed in parentheses
- separated by commas.

36

Positional Parameters: Invocation

General form of a **macro call** with positional parameters:

```
%macro-name (value-1, ... value-n)
```

Positional parameter **values** are

- enclosed in parentheses
- separated by commas
- specified in the same order as in the macro definition.

Parameter values can be any text. The default value for a positional parameter is a *null value*.

If a macro definition declares parameters, the parentheses are required in the corresponding macro call.

37

Initial Macro Resolution with Parameters

The macro call resolves to the text stored in the macro definition. Several macro parameter references are stored as part of the macro definition.

Your

Submitted Code `%printdsn(sashelp.class,name age)`

WORK.SASMACR.PRINTDSN.MACRO

```
proc print data=&dsn;
  var &vars;
  title1 "Data from &dsn";
run;
```

Macro Processor

Initial macro resolution

```
proc print data=&dsn;
  var &vars;
  title1 "Data from &dsn";
run;
```

38

Resolution of Macro Parameters

The macro parameters in the macro definition are resolved before the code it sent to the SAS compiler.

Initial macro resolution

```
proc print data=&dsn;
  var &vars;
  title1 "Data from &dsn";
run;
```

Local Symbol Table

DSN	sashelp.class
VAR	name age

Macro Processor

Code to be executed

```
proc print data=sashelp.class;
  var name age;
  title1 "Data from sashelp.class";
run;
```

39

Keyword Macro Parameters

Keyword parameters permit the declaration of default parameter values.

The name/value pair for a keyword parameter

- assigns the default value for the macro definition
- **overrides** the default value for a specific macro call.

```
%macro printdsn(dsn=&syslast,vars=_all_);
  proc print data=&dsn;
    var &vars;
    title1 "Data from &dsn";
  run;
%mend printdsn;

%printdsn(dsn=sashelp.class,vars=name age)
```

The diagram shows a macro definition and a macro call. In the definition, `dsn=&syslast` and `vars=_all_` are highlighted in blue and red respectively. In the call, `dsn=sashelp.class` and `vars=name age` are highlighted in blue and red respectively. A blue arrow points from the `dsn` parameter in the call to the `dsn` parameter in the definition. A red arrow points from the `vars` parameter in the call to the `vars` parameter in the definition.

40

Keyword Parameters: Definition

General form of a **macro definition** with a keyword parameter list :

```
%MACRO macro-name (keyword=value, ..., keyword=value);
  macro text
%MEND <macro-name>;
```

Keyword parameter declarations are

- enclosed in parentheses
- separated by commas
- name/value pairs delimited with an equal (=) sign.

41

Keyword Parameters: Invocation

General form of a **macro call** with keyword parameters:

```
%macro-name (keyword=value, ..., keyword=value)
```

Like positional parameters, keyword parameter values are

- enclosed in parentheses
- separated by commas.

Unlike positional parameters, keyword parameter values

- can be specified in any order
- are specified using a complete name/value pair.

42

Keyword Parameters: Using Defaults

To use the default value for a keyword parameter, omit the parameter from the macro call.

```
%printdsn(dsn=sashelp.class)
```

To use all default values, omit all parameters, ending the macro call with empty parentheses.

```
%printdsn()
```

43

Calling the Macro using the Defaults Values

The DATA step creates **work.class** and sets the value for the automatic macro variable **SYSLAST**. The PRINTDSN macro is called using the default values.

Your
Submitted
Code

```
data class;
  set sashelp.class;
run;
%printdsn()
```

WORK.SASMACR.PRINTDSN.MACRO

```
proc print data=&dsn;
  var &vars;
  title1 "Data from &dsn";
run;
```

Macro Processor

Initial macro
resolution

```
proc print data=&dsn;
  var &vars;
  title1 "Data from &dsn";
run;
```

44

Parameter Resolution: Keyword Defaults

The references to macro parameters in the macro definition text are resolved based on parameter settings.

Initial macro
resolution

```
proc print data=&dsn;
  var &vars;
  title1 "Data from &dsn";
run;
```

Local Symbol Table

DSN	&syslast
VAR	_all_

Macro Processor

Code after
initial parameter
resolution

```
proc print data=&syslast;
  var _all_;
  title1 "Data from &syslast";
run;
```

45

Further Resolution of Macro Variables

All remaining macro variable references are resolved.

Code after
initial parameter
resolution

```
proc print data=&syslast;
  var _all_;
  title1 "Data from &syslast";
run;
```

Global Symbol Table

SYSLAST	work.class
---------	------------

Local Symbol Table

DSN	&syslast
VARS	_all_

Macro Processor

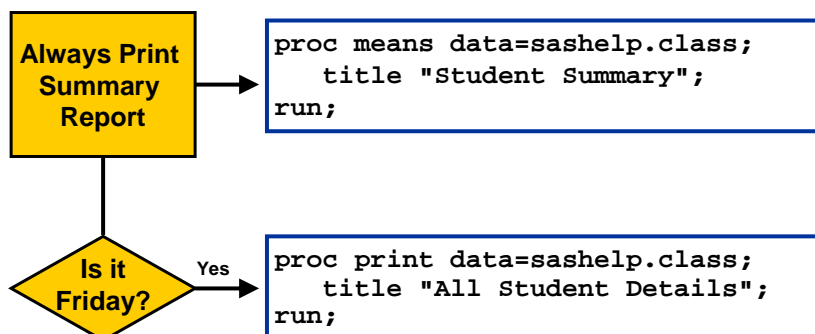
Code to be
executed

```
proc print data=work.class;
  var _all_;
  title1 "Data from work.class";
run;
```

46

The Need for Conditional Macro Programming

Suppose we have two portions of SAS program code that must be run at different times.



47

Conditional Macro Processing

You can perform conditional execution with %IF-%THEN and %ELSE statements.

General form of %IF-%THEN and %ELSE statements:

```
%IF expression %THEN text,  
%ELSE text,
```

The %ELSE statement is optional.

These macro language statements can only be used inside a macro definition.

48

Macro Comparison Expressions

Macro language comparison expressions are similar to DATA step expressions in that they support

- the comparison operators EQ, LT, LE, GT, GE, and NE
- their symbolic equivalents =, ^=, <, <=, >, and >=

Macro comparison expressions differ from DATA step expressions in that they

- reference macro variables, not DATA step variables
- do not support ranges in the form **1 <= &x <= 10**
- do not support the IN operator
- do not support special WHERE operators

49

In SAS 9.2, the Macro facility will support the IN operator.

Conditional Processing: %DO-%END

The text following %THEN and %ELSE can be

- a single macro programming statement
- any text that does not contain a semicolon.

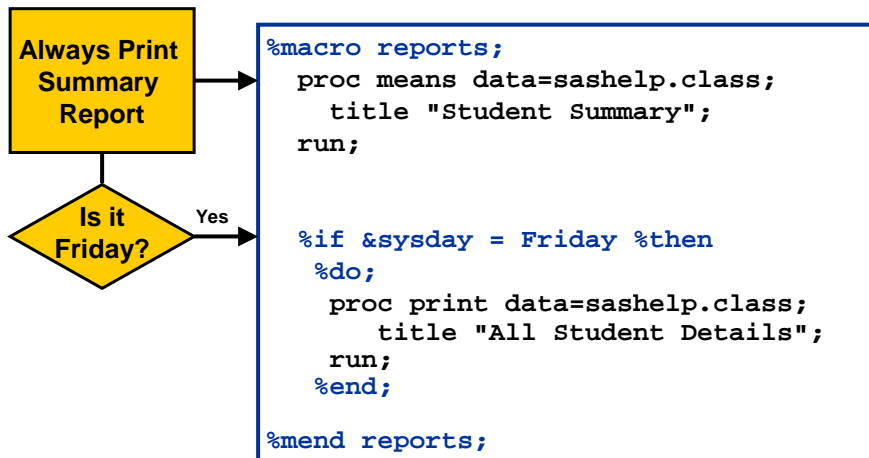
When the text contains semicolons, use a %DO - %END group following %THEN or %ELSE.

```
%IF expression %THEN %DO;  
    statement, statement,...  
%END;  
%ELSE %DO;  
    statement, statement,...  
%END;
```

50

Conditional Macro Logic

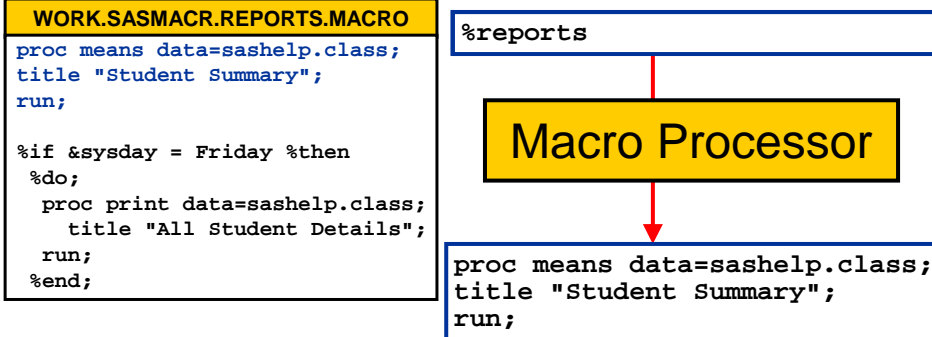
Example: Implement a macro definition with the %IF-%THEN statement and a %DO-%END group.



51

Conditional Macro Logic

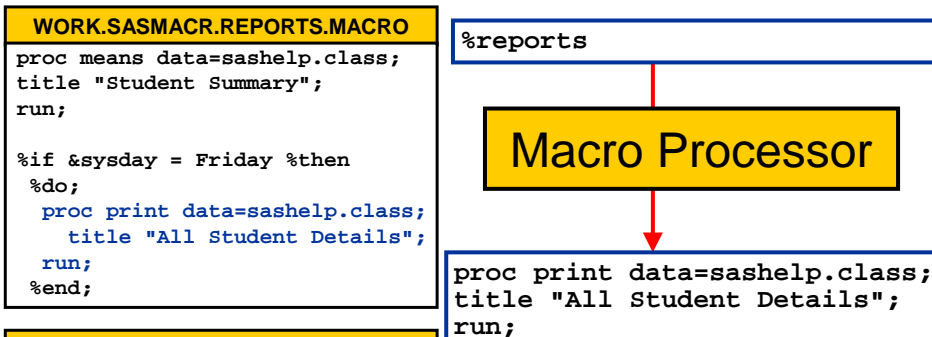
The first portion of the macro text is not controlled by conditional logic, so it is executed for all macro calls.



52

Conditional Macro Logic

Because the second portion of the macro text is controlled by conditional logic, it is executed only when the %IF statement expression is true.



Global Symbol Table	
SYSDAY	Friday

53

Iterative Macro Processing

Many macro applications require iterative processing.

The iterative %DO statement can repeatedly

- execute macro language statements
- generate SAS code.

General form of the iterative %DO statement:

```
%DO index-variable=start %TO stop <%BY increment>;  
  text  
%END;
```

%DO and %END statements are valid only inside a macro definition.

54

Iterative %DO Statement: Rules

- Index-variable is a macro variable.
- Index-variable is created in the local symbol table if it does not already exist in an existing symbol table.
- Start, stop, and increment values can be any valid macro expressions that resolve to integers.
- %BY clause is optional (default increment is 1).

55

Generating Complete Steps: Macro

Example: Iteratively generate complete SAS steps.

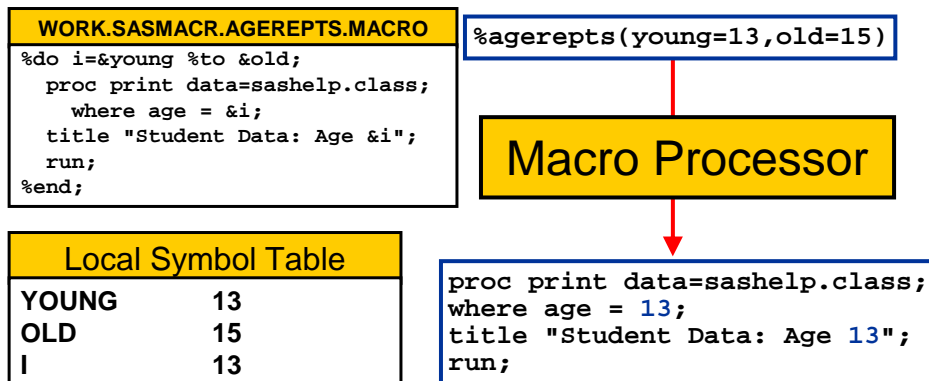
```
%macro agerepts(young=11,old=19);
  %do i=&young %to &old;
    proc print data=sashelp.class;
      where age = &i;
      title "Student Data: Age &i";
    run;
  %end;
%mend agerepts;

%agerepts(young=13,old=15)
```

56

Iterative Generation of SAS Code: Iteration 1

The first iteration of the %DO loop sets the macro variable AGE to 13. The complete macro resolution generates the first PROC PRINT step.



57

Iterative Generation of SAS Code: Iteration 2

The %DO loop index variable increments to 14. The second PROC PRINT step is generated.

WORK.SASMACR.AGEREPTS.MACRO

```
%do i=&young %to &old;
  proc print data=sashelp.class;
    where age = &i;
    title "Student Data: Age &i";
  run;
%end;
```

```
%agerepts(young=13,old=15)
```

Macro Processor

Local Symbol Table

YOUNG	13
OLD	15
I	14

```
proc print data=sashelp.class;
  where age = 14;
  title "Student Data: Age 14";
run;
```

58

Iterative Generation of SAS Code: Iteration 3

The %DO loop index variable increments to 15. The final PROC PRINT step is generated.

WORK.SASMACR.AGEREPTS.MACRO

```
%do i=&young %to &old;
  proc print data=sashelp.class;
    where age = &i;
    title "Student Data: Age &i";
  run;
%end;
```

```
%agerepts(young=13,old=15)
```

Macro Processor

Local Symbol Table

YOUNG	13
OLD	15
I	15

```
proc print data=sashelp.class;
  where age = 15;
  title "Student Data: Age 15";
run;
```

59

Generating Complete Steps: Resulting Code

SAS code generated by the macro call

```
proc print data=sashelp.class;
  where age = 13;
  title "Student Date: Age 13";
run;
proc print data=sashelp.class;
  where age = 14;
  title "Student Date: Age 14";
run;
proc print data=sashelp.class;
  where age = 15;
  title "Student Date: Age 15";
run;
```

60

Where to Go Next

- SAS® Macro Language – 2 days
- SAS® Macro Programming: Advanced Topics - 1 day

<http://support.sas.com/training/>

61