

Getting started ○○○○	Single variable insertion ○○○○○○○○	Macro modules ○○○○○○○○○○○○
-------------------------	---------------------------------------	-------------------------------

# SAS Macros

Shannon Pileggi

STAT 330

STAT 330: Lecture 5
1 / 25

Getting started ○○○○	Single variable insertion ○○○○○○○○	Macro modules ○○○○○○○○○○○○
-------------------------	---------------------------------------	-------------------------------

## OUTLINE

Getting started

Single variable insertion

Macro modules

STAT 330: Lecture 5
2 / 25

Getting started ●○○○	Single variable insertion ○○○○○○○○	Macro modules ○○○○○○○○○○○○
-------------------------	---------------------------------------	-------------------------------

## SAS macros

- ▶ The SAS macro facility is a *text processing facility*
- ▶ It allows us to insert/include line(s) of SAS code anywhere in the entire program
- ▶ This provides a **very** convenient way to automate many processes
- ▶ It is very much like having a handy recording of SAS code which you can play back whenever you need
- ▶ Macros can be broken down into two main types:
  - ▶ single variable insertion
  - ▶ multiple lines insertion

STAT 330: Lecture 5
3 / 25

Getting started ●●○○	Single variable insertion ○○○○○○○○	Macro modules ○○○○○○○○○○○○
-------------------------	---------------------------------------	-------------------------------

## The Macro Processor and the Standard SAS Processor

- ▶ In the presence of macro code, SAS will go through an initial scan of your code and 'resolve' any macros first.
- ▶ After the initial scan, the appropriate line(s) of SAS code are 'inserted' (it's like a program that writes a program)
- ▶ Finally, SAS compiles the full code and executes as usual

STAT 330: Lecture 5
4 / 25

## Macro triggers

When a SAS program is submitted, two token sequences are recognized as *macro triggers*:

1. `&name-token` - a macro variable reference
2. `%name-token` - a macro statement, function, or call

(A *token* is a fundamental unit of text.)

## Debugging macros

Items that are underlined represent the default SAS settings:

- ▶ MERROR | NOMERROR – issues a warning in the log window when attempting to invoke a macro that does not exist.
- ▶ SERROR | NOSERROR – issues a warning in the log window when attempting to use a macro variable that does not exist.
- ▶ MLOGIC | NOMLOGIC – prints (in the log window) details of every macro step.
- ▶ MPRINT | NOMPRINT – prints (in the log window) details of what SAS ultimately “sees” during the Standard SAS Processor stage.
- ▶ SYMBOLGEN | NOSYMBOLGEN – prints (in the log window) the resolved values of any macro variables.

Use the following to ensure ALL your macro tools are made available to you:

```
OPTIONS MPRINT MLOGIC SYMBOLGEN;
```

Getting started

Single variable insertion

Macro modules

## Macro Variable: The Single Variable Insertion

- ▶ Macro variables have a *single* value and do *not* belong to a data set
- ▶ When reference, macro variable names are prefixed with an ampersand (&)
- ▶ All macro variables are stored as *character* based variables
- ▶ You may name a macro variable whatever you wish, but do not use `sys` as the first three letters of a macro variable. Such variables are reserved for special purposes.
- ▶ A macro variable may have a **global scope** (can be used anywhere in the code) or a **local scope** (used only in a macro).

## Automatic macro variables

- ▶ SAS has automatic macro variables that begin with the prefix `sys`
- ▶ <http://support.sas.com/documentation/cdl/en/mcrolref/61885/HTML/default/viewer.htm#a003167023.htm>

SAS Code

```
TITLE "Contents of Baseball Data on &sysdate9" ;  
PROC CONTENTS DATA = sashelp.baseball VARNUM ;  
RUN ;  
TITLE ;
```

SAS Code

## Macro Variable: The Single Variable Insertion

- ▶ To create a basic macro variable we use  
`%LET macro_variable_name = value ;`
- ▶ %LET statements are valid in open code (any where in SAS program)
- ▶ When *assigning* a macro variable a value
  - ▶ do not do `%LET &macro_variable_ = value ;`
  - ▶ do not put quotes around the *value*
- ▶ This is useful for changing values during a SAS program without having to change the entire program itself
- ▶ To use the macro variable that you've created call it with `&macro_variable_name`

## Resolving macro variables

SAS Code

```
%LET my_GPA = 3.3;  
%LET country = New Zealand;
```

SAS Code

**SAS Code**`IF GPA = &my_GPA;`**Resolves to**`IF GPA = 3.3;`**SAS Code**`title "Addresses in &country";`**Resolves to**`title "Addresses in New Zealand";`

## Baseball data

SAS Code

```
TITLE "Data = sashelp.basesball, Obs = 10" ;  
PROC PRINT DATA = sashelp.baseball (OBS = 10);  
RUN ;  
TITLE ;
```

SAS Code

**On your own:** Convert the dataset name of `sashelp.basesball` and the number of observations printed `10` to macro variables named `dsn` and `num`.

Getting started  
0000

Single variable insertion  
000000●

Macro modules  
0000000000

SAS Code

```
%let x=15;
%let y=10;
%let z=&x-&y;
```

SAS Code

What is the value of the SAS macro variable z?

1. 5
2. 15-10
3. x-y
4. &15-&10
5. error

Getting started  
0000

Single variable insertion  
000000●

Macro modules  
0000000000

## Macro variables for path names

Original Code

```
LIBNAME mylib "X:/spileggi/Data Sets/" ;
PROC IMPORT OUT = mylib.babies
  DATAFILE = "X:/spileggi/Data Sets/babies.csv"
  DBMS = CSV REPLACE;
RUN;
```

Original Code

New Code

```
%LET mypath = X:/spileggi/Data Sets/ ;
LIBNAME mylib "&mypath" ;
PROC IMPORT OUT = mylib.babies
  DATAFILE = "&mypath.babies.csv"
  DBMS = CSV REPLACE;
RUN;
```

New Code

A period allows you to concatenate a macro variable with other text.

Getting started  
0000

Single variable insertion  
00000000

Macro modules  
●0000000000

Getting started

Single variable insertion

Macro modules

Getting started  
0000

Single variable insertion  
00000000

Macro modules  
●0000000000

## Macro Modules: Multiple Lines Insertion

If you ever find yourself writing the same code over and over you should consider using a macro module.

Macro Definition

```
%MACRO macro_name ;

  ...code...

%MEND ;
```

Macro Definition

Macro Execution

```
%macro_name ;
```

Macro Execution

## Macro module, no parameters

### Macro Definition

```
%MACRO myprint ;
TITLE "DATA = &dsn, OBS = &num" ;
PROC PRINT DATA = &dsn (OBS=&num);
RUN ;
TITLE ;
%MEND ;
```

### Macro Definition

### Macro Execution

```
%LET num = 10 ;
%LET dsn = sashelp.baseball ;

%myprint ;
```

### Macro Execution

## Macro module, positional parameters

### Macro Definition

```
%MACRO myprint(dsn, num) ;
TITLE "DATA = &dsn, OBS = &num" ;
PROC PRINT DATA = &dsn (OBS=&num);
RUN ;
TITLE ;
%MEND ;
```

### Macro Definition

### Macro Executions

```
%myprint(sashelp.baseball,5) ;
%myprint(sashelp.class,3) ;
```

### Macro Executions

- ▶ no equal sign in MACRO definition
- ▶ the parameter values match the order in which they are listed in the macro definition
- ▶ the order of the parameter values matters

## Macro module, keyword parameters

### Macro Definition

```
%MACRO myprint(dsn = sashelp.baseball, num = 5) ;
TITLE "DATA = &dsn, OBS = &num" ;
PROC PRINT DATA = &dsn (OBS=&num);
RUN ;
TITLE ;
%MEND ;
```

### Macro Definition

### Macro Executions

```
%myprint ;
%myprint();
%myprint(dsn = sashelp.baseball, num = 5) ;
%myprint(num = 5, dsn = sashelp.baseball) ;
%myprint(num = 3) ;
%myprint(dsn = sashelp.class, num = 3) ;
```

- ▶ uses an equal sign in MACRO definition
- ▶ sets default values for parameters
- ▶ can replace all or some subset of default values
- ▶ the order of the parameter values matter does not matter

## Developing Macro Applications

Follow these steps to create and de-bug your SAS macros:

1. Write and debug a SAS program without macro coding.
2. Generalize the program by replacing hardcoded values with macro variable references.
3. Create a macro definition with macro parameters.
4. Add macro-level programming for conditional and iterative processing.

## Tips and warnings

### Tips:

- ▶ ALWAYS include the macro debugging options in your SAS program when writing macros

```
OPTIONS MPRINT MLOGIC SYMBOLGEN;
```

- ▶ With these options, you should be able to see what values a macro parameter resolves to. Another way is with %PUT, which prints text to the LOG.

```
%PUT dsn = &dsn , num = &num;
```

### Warnings:

- ▶ use the `\* ... *\` commenting style when coding macros
- ▶ use double quotations (instead of single quotations) when calling macro variable names

```
%LET month = January;
```

Which of the following produces the title  
*The month is January?*

1. title "The month is &month";
2. title 'The month is &month';
3. title "The month is %month";
4. title 'The month is %month';

## Macro Conditional Logic

- ▶ We can use conditional logic **outside of data steps** within macros using %IF, %THEN, %DO -- %END, %ELSE
- ▶ These statements work like their counterparts IF, THEN, DO -- END, ELSE
- ▶ These conditional logic statements
  - ▶ **can only be used within a macro module**
  - ▶ are 'seen' only during the initial macro resolution scanning process
  - ▶ are NOT included into the SAS code itself

## %DO loop

- ▶ The %DO ... %TO statement allows you to perform loops inside a macro.
- ▶ %DO loops in macros have the same kind of structure as standard DO loops in regular SAS code.
- ▶ As with the conditional logic statements in macros [i.e. %IF ... %THEN], these statements **must** be embedded within a macro module – they **cannot** be placed outside of a macro module and in "open SAS code".

## Using CALL SYMPUTX

Using CALL SYMPUTX allows you to take a value from the data step and assign it to a macro variable.

```
CALL SYMPUTX("macrovariablename" ,value)
```

- ▶ macrovariablename **must** be surrounded by quotes
- ▶ value can be
  - ▶ a string in quotes (character or numeric)
  - ▶ the name of a variable that SAS will use to assign a value (in this case, do NOT use quotes)