From Muggles to Macros

Transfiguring Your SAS® Programs With Dynamic, Data-Driven Wizardry

Josh Horstman, Nested Loop Consulting
Richann Watson, DataRich Consulting

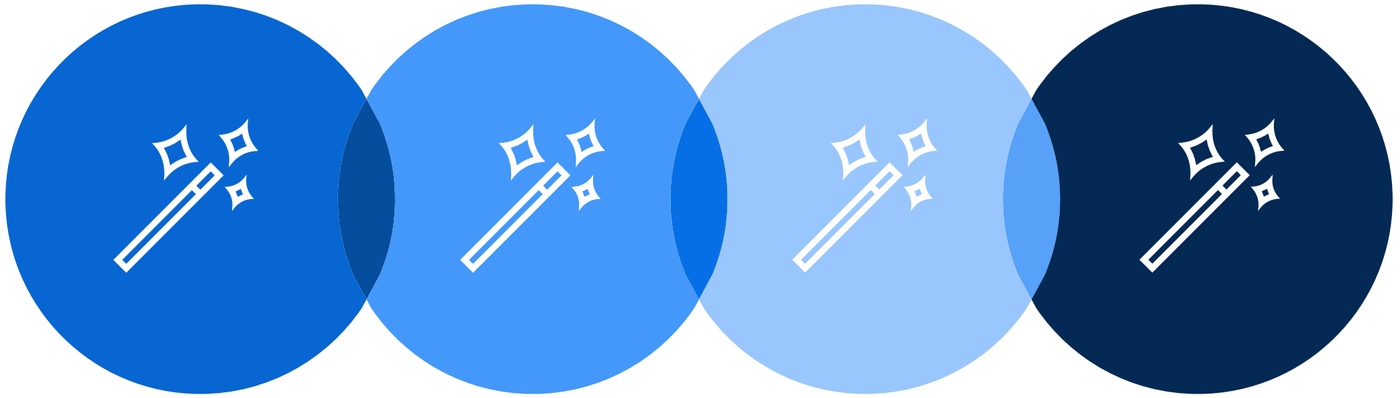
sas innovate 2024

Introduction

- Static “muggle” code is full of hardcodes and data dependencies
  - Not flexible: Breaks easily when unexpected inputs or conditions are present
  - Difficult to maintain: Modifications needed when data or environment changes
  - Difficult to reuse: Modifications needed to use for another project

- Macro Language “magic” can eliminate these problems!
  - Dynamic: Code automatically adapts to changing inputs and conditions
  - Data-Driven: Programming logic is controlled by the data and requires little maintenance
  - Reusable: Code can easily be used in a variety of situations with little to no modification

Overview



1 Macro Language Review
The Basic Spellbook

2 Applying Macro Magic to Data Values
Spell #1

3 Applying Macro Magic to Metadata
Spell #2

4 Applying Macro Magic to Environmental Data
Spell #3

sas innovate

Macro Language Review

The Basic Spellbook

sas innovate

Macro Processing: A Simplified Overview

- When a SAS program is submitted:
  - Word scanner parses statements into tokens.
  - Tokens are sent to compiler for syntax checking.
  - Execution occurs when step boundary is reached.
- If the word scanner detects macro triggers (% or &):
  - Macro elements routed to macro processor.
  - Macro variables resolved and macro statements executed.
  - Output from macro processor must be rescanned for additional macro language elements.

Macro Processing: A Visual Guide

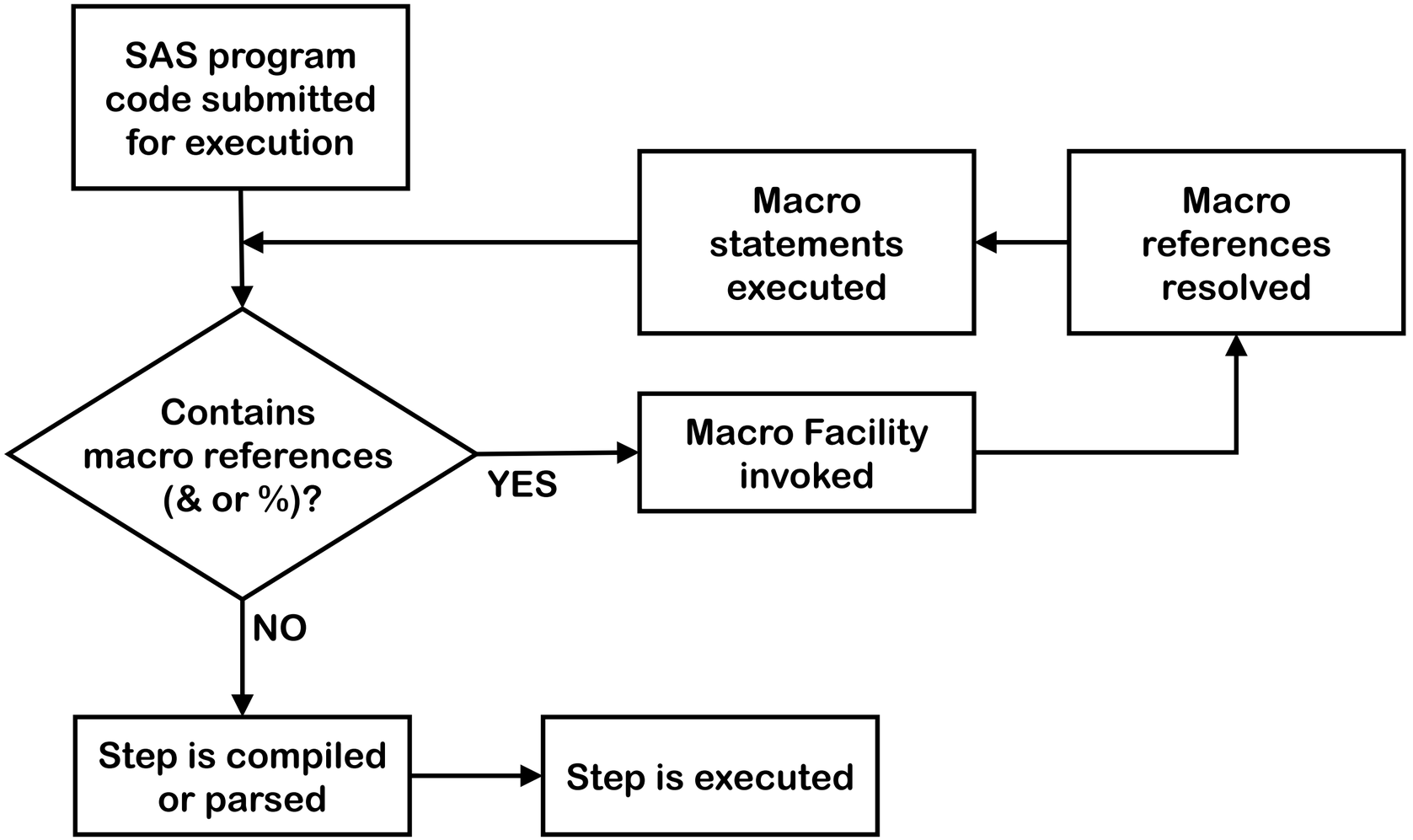


Diagram from Carpenter's Complete Guide to the SAS® Macro Language, 3rd Edition. Used with permission.

sas innovate

Creating Macro Variables using %LET

- Assigning a value to a macro variable:

```
%let output_path = C:\temp;
```

- Subsequent references to macro variable replaced with value by macro processor:

```
filename myfile "&output_path\myfile.txt";
```

- becomes

```
filename myfile "C:\temp\myfile.txt";
```

Limitations of %LET

VIEWTABLE: Sashelp.Class (Student Data)

	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5
2	Alice	F	13	56.5	84

- Macro processor assigns value before SAS code executes.

```
data _null_;
  set sashelp.class;
  where name='Alfred';
  %let alfred_age = age;
run;
```

This %LET statement will not have the desired effect.

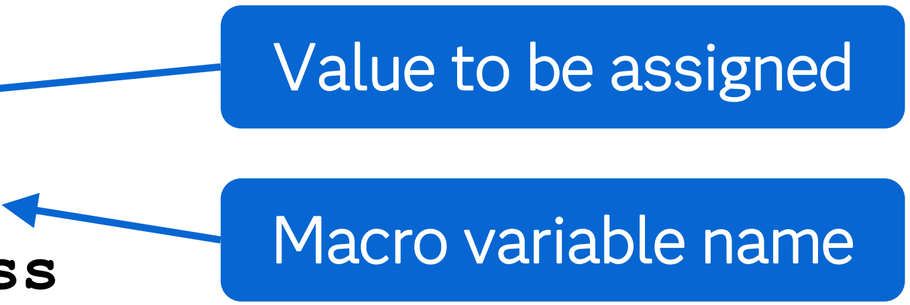
- Macro variable **alfred_age** is literally assigned the value "**age**".
- SAS compiler only sees this:

```
data _null_;
  set sashelp.class;
  where name='Alfred';
run;
```

Creating Macro Variables at Execution Time using the SQL Procedure

- INTO clause assigns macro variable values during PROC SQL:

```
proc sql noprint;
  select age
    into :alfred_age
    from sashelp.class
    where name='Alfred';
quit;
```



- Macro variable **alfred_age** will be assigned the value "**14**".

Macro Processing: Timing is Everything!

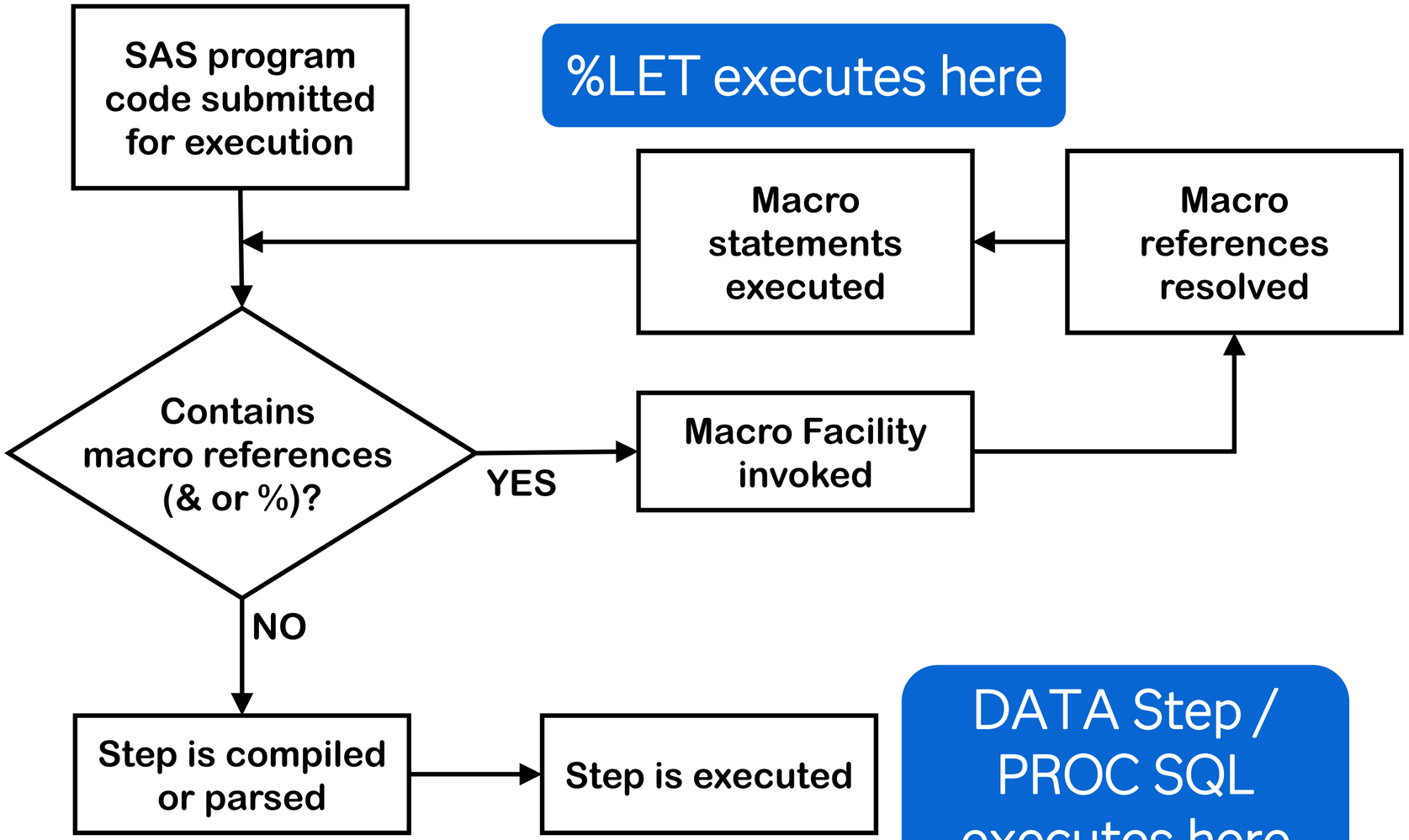


Diagram from Carpenter's Complete Guide to the SAS® Macro Language, 3rd Edition. Used with permission.

sas innovate

Applying Macro Magic to Data Values

Spell #1

sas innovate

Incantation #1: The Macro Variable List

- Macro Variable List – a series of macro variables, each storing one value
- Named with a common prefix and sequential suffix to enable processing in a loop
- Example: A macro variable list containing the unique values of the ORIGIN variable from the SASHELP.CARS data set

```
%let origin1 = Asia;
%let origin2 = Europe;
%let origin3 = USA;
```

- But we want to create these dynamically, not by hard-coding!
- Must be created at execution time to have access to data values.

Creating a Macro Variable List using PROC SQL

```
proc sql noprint;
   select distinct origin
     into :origin1-
     from sashelp.cars
      order by origin;
   %let numorigins = &sqlobs;
quit;
```

Get only unique values of ORIGIN.

The dash creates a series of sequential macro variables, one for each value returned by the query. No need for an upper bound.

Create one more macro variable so we know how many items are in our list.

```
11    %put _user_;
GLOBAL NUMORIGINS 3
GLOBAL ORIGIN1 Asia
GLOBAL ORIGIN2 Europe
GLOBAL ORIGIN3 USA
```

Using Macro Variable Lists

Part 1

- Access individual list elements using macro variable reference:

  &origin1 → Resolves to: **Asia**
  &origin2 → Resolves to: **Europe**
  &origin3 → Resolves to: **USA**

- Cannot use **&origin&i**
- Macro processor interprets this as two macro variable references:
- Macro variable **origin** does not exist.

```
11   %put _user_;
GLOBAL NUMORIGINS 3
GLOBAL ORIGIN1 Asia
GLOBAL ORIGIN2 Europe
GLOBAL ORIGIN3 USA
```

Using Macro Variable Lists

Part 2

- Instead, use **&&origin&i**.
- Use in a loop:

```
%do i = 1 %to &numorigins;
  %put Item &i: &&origin&i;
%end;
```

Original: **&&origin&i**
1st pass: **&origin1** (&& resolves to &, origin is just text, &i resolves to 1)
2nd pass: **Asia** (resolved value of macro variable origin1)

```
Item 1: Asia
Item 2: Europe
Item 3: USA
```

Example #1: Dynamic Report Creation

- Goal: Create a separate plot in a separate PDF output file for each unique value of STOCK in the SASHELP.STOCKS data set.

- Muggle approach: Code a separate call to PROC SGPLOT for each unique value of STOCK

- Macro Wizard approach: Use a macro variable list to dynamically generate the calls to PROC SGPLOT.

Example #1: Dynamic Report Creation

Muggle Code

```
ods pdf file="IBM.pdf";
  proc sgplot data=sashelp.stocks;
    where stock = "IBM";
    highlow x=date high=high low=low;
  run;
ods pdf close;
```

```
ods pdf file="Intel.pdf";
  proc sgplot data=sashelp.stocks;
    where stock = "Intel";
    highlow x=date high=high low=low;
  run;
ods pdf close;
```

```
ods pdf file="Microsoft.pdf";
  proc sgplot data=sashelp.stocks;
    where stock = "Microsoft";
    highlow x=date high=high low=low;
  run;
ods pdf close;
```

This code must be repeated for each unique value of STOCK.

Example #1: Dynamic Report Creation

Macro Wizard Code – Part 1 of 2

```
%macro graph_stocks;

  proc sql noprint;
    select distinct stock
      into :stock1-
      from sashelp.stocks;
    %let numstocks = &sqlobs;
  quit;
```

Use PROC SQL to place the unique values of STOCK in a macro variable list.

```
GRAPH_STOCKS NUMSTOCKS 3
GRAPH_STOCKS STOCK1 IBM
GRAPH_STOCKS STOCK2 Intel
GRAPH_STOCKS STOCK3 Microsoft
```

Example #1: Dynamic Report Creation

Macro Wizard Code – Part 2 of 2

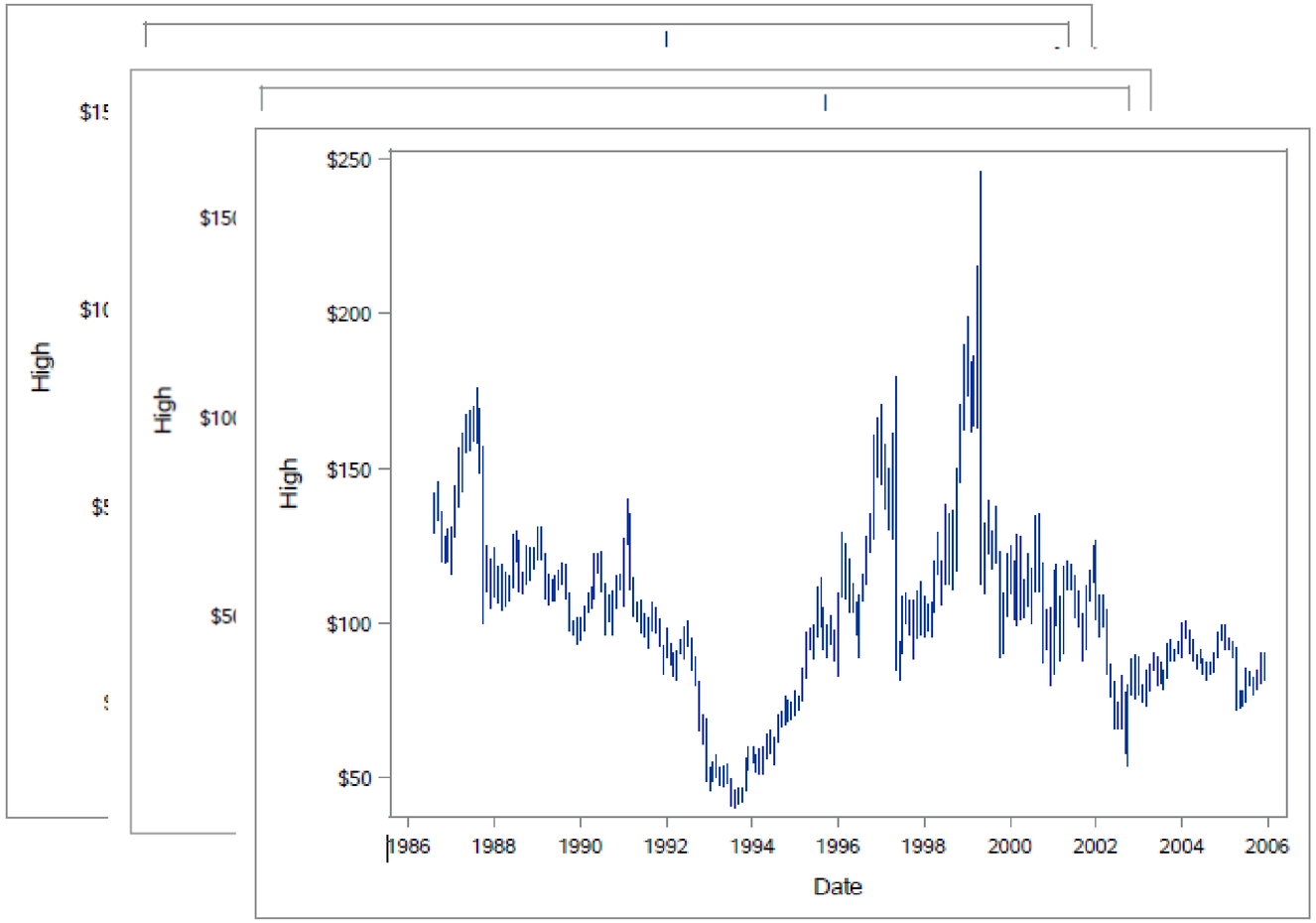
```
%do i = 1 %to &numstocks;
  ods pdf file="&&stock&i...pdf";
    proc sgplot data=sashelp.stocks;
      where stock = "&&stock&i";
      highlow x=date high=high low=low;
    run;
  ods pdf close;
%end;
%mend graph_stocks;

%graph_stocks
```

Each iteration of the %DO loop generates the code to plot one stock.

Two-pass macro variable resolution

Example #1: Dynamic Report Creation

Output



IBM.pdf Intel.pdf Microsoft.pdf

A separate PDF file for each unique value of STOCK

sas innovate

Example #2: Building Variable Attributes

- Goal: Modify SASHELP.CLASS so variable attributes and order conform to desired specifications.
- Muggle approach: Hardcode all attributes using ATTRIB statements.
- Macro Wizard approach:
  - Specify desired attributes in a data set.
  - Use macro variables lists to dynamically build ATTRIB statements from data set.

VIEWTABLE: Sashelp.Class (Student Data)

	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5
2	Alice	F	13	56.5	84

Example #2: Building Variable Attributes

CURRENT ORDER AND ATTRIBUTES

Variables in Creation Order			
#	Variable	Type	Len
1	Name	Char	8
2	Sex	Char	1
3	Age	Num	8
4	Height	Num	8
5	Weight	Num	8

DESIRED ORDER AND ATTRIBUTES

POS	VARIABLE	LABEL	TYPE	LEN	FORMAT
1	NAME	Student Name	Char	7	
2	AGE	Age	Num	3	
3	SEX	Gender	Char	1	
4	HEIGHT	Height	Num	8	8.2
5	WEIGHT	Weight	Num	8	8.2
6	BMI	Body Mass Index	Num	8	8.2

Example #2: Building Variable Attributes

Muggle Code

```
options varlenchk = NOWARN;
data myclass;
    attrib name length = $7 label = 'student name';
    attrib age length = 3 label = 'age';
    attrib sex length = $1 label = 'sex';
    attrib height length = 8 label = 'height' format = 8.2;
    attrib weight length = 8 label = 'weight' format = 8.2;
    attrib bmi length = 8 label = 'body mass index'
           format = 8.2;
    set sashelp.class;
    call missing(bmi);
run;
```

Setting the VARLENCHK option to NOWARN avoids a log warning.

```
WARNING: Multiple lengths were specified
for the variable NAME by input data set(s).
This can cause truncation of data.
```

Example #2: Building Variable Attributes

Macro Wizard Code – Part 1 of 2

- Use PROC SQL to create a number of macro variables that will capture all the various attributes

```
proc sql noprint;
    select variable, label,
           type, len, format
        into :var1 - ,
             :lbl1 - ,
             :typ1 - ,
             :len1 - ,
             :fmt1 -
    from attrs;
    %let numvars = &sqlobs;
quit;
```

Select variables that will be used to create the macro variable lists.

Specify names of macro variable lists in same order as selected variables.

MACRO VARIABLE	VALUE
FMT1	
…	
FMT6	8.2
LBL1	Student Name
…	
LBL6	Body Mass Index
LEN1	7
…	
LEN6	8
TYP1	Char
…	
TYP6	Num
VAR1	NAME
…	
VAR6	BMI
NUMVARS	6

Example #2: Building Variable Attributes

Macro Wizard Code – Part 2 of 2

ATTRIB must precede SET to correctly assign attributed in PDV.

```
%macro attrib(dsn = );
   data myfile;
      %do i = 1 %to &numvars;
      attrib &&var&i
         %if &&len&i ne %then %do;
            %if &&typ&i = Char %then length = $&&len&i;
            %else length = &&len&i;
         %end;
         %if &&fmt&i ne %then format = &&fmt&i;
         %if &&lbl&i ne %then label = "&&lbl&i";
         ;
      %end;
      call missing(of _all_);
      set &dsn;
   run;
%mend attrib;
%attrib(dsn = SASHELP.CLASS);
```

%DO loop generates series of ATTRIB statements.

Ends ATTRIB statement

Initializes all variables in PDV to missing

Example #2: Building Variable Attributes

Output

MPRINT option displays SAS code generated by the macro language.

```
1          %attrib(dsn = SASHELP.CLASS);
MPRINT(ATTRIB):   data myfile;
MPRINT(ATTRIB):   attrib NAME length = $7 label = "Student Name" ;
MPRINT(ATTRIB):   attrib AGE length = 3 label = "Age" ;
MPRINT(ATTRIB):   attrib SEX length = $1 label = "Sex" ;
MPRINT(ATTRIB):   attrib HEIGHT length = 8 format = 8.2 label = "Height" ;
MPRINT(ATTRIB):   attrib WEIGHT length = 8 format = 8.2 label = "Weight" ;
MPRINT(ATTRIB):   attrib BMI length = 8 format = 8.2 label = "Body Mass Index" ;
MPRINT(ATTRIB):   call missing(of _all_);
MPRINT(ATTRIB):   set SASHELP.CLASS;
MPRINT(ATTRIB):   run;
```

```
proc contents data = myfile varnum;
run;
```

Use PROC CONTENTS to verify correct attributes were assigned.

Variables in Creation Order

#	Variable	Type	Len	Format	Label
1	NAME	Char	10		Student Name
2	AGE	Num	3		Age
3	SEX	Char	1		Sex
4	HEIGHT	Num	8	8.2	Height
5	WEIGHT	Num	8	8.2	Weight
6	BMI	Num	8	8.2	Body Mass Index

Applying Macro Magic to Metadata

Spell #2

sas innovate

Incantation #2: Automatic Macro Variables

- Several automatic macro variables are created when a SAS session starts.
- Some can be quite useful for dynamic programming:

Macro variable	Description	Sample value
SYSDATE9	Current date in DATE9 format	17APR2024
SYSERR	Return code status from last step executed	0
SYSLAST	Name of last SAS data set created/modified	WORK.CLASS
SYSNOBS	Number of observations in last data set created/modified	19
SYSSCP	Identifier of the current operating system	WIN
SYSUSERID	System ID of current user	rwatson

- To write current values of all automatic macro variables to the log:

```
%put _automatic_;
```

Example #3: Process Only Non-Empty Data sets

Macro Wizard Code

- Goal: Confirm that a subset of a data set contains observations before further processing

- Muggle approach: Don’t check first, just execute the code even with no observations.

- Macro Wizard approach: Use conditional macro logic to run a certain portion of code only if the data subset contains a nonzero number of observations.

Example #3: Process Only Non-Empty Data sets

Macro Wizard Code

```
%macro myreport(indsn=,subset=);

      data reportdata;
            set &indsn;
            where &subset;
      run;

      %if &sysnobs ne 0 %then %do;
            proc print data=reportdata;
            run;
      %end;
      %else %put NOTE: The specified subset is empty.;

%mend myreport;
```

&SYSNOBS contains number of observations in REPORTDATA

Example #3: Process Only Non-Empty Data sets

Output

```
%myreport(
   indsn   = sashelp.class,
   subset = %str(age=12)
   )
```

Obs	Name	Sex	Age	Height	Weight
1	James	M	12	57.3	83.0
2	Jane	F	12	59.8	84.5
3	John	M	12	59.0	99.5
4	Louise	F	12	56.3	77.0
5	Robert	M	12	64.8	128.0

```
%myreport(
   indsn   = sashelp.class,
   subset = %str(age=17)
   )
```

```
NOTE: There were 5 observations read from the data set SASHELP.CLASS.
      WHERE age=12;
NOTE: The data set WORK.REPORTDATA has 5 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.01 seconds


NOTE: There were 5 observations read from the data set WORK.REPORTDATA.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.02 seconds
      cpu time            0.00 seconds
```

```
NOTE: There were 0 observations read from the data set SASHELP.CLASS.
      WHERE age=17;
NOTE: The data set WORK.REPORTDATA has 0 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds


NOTE: The specified subset is empty.
```

Incantation #3: Dictionary Tables

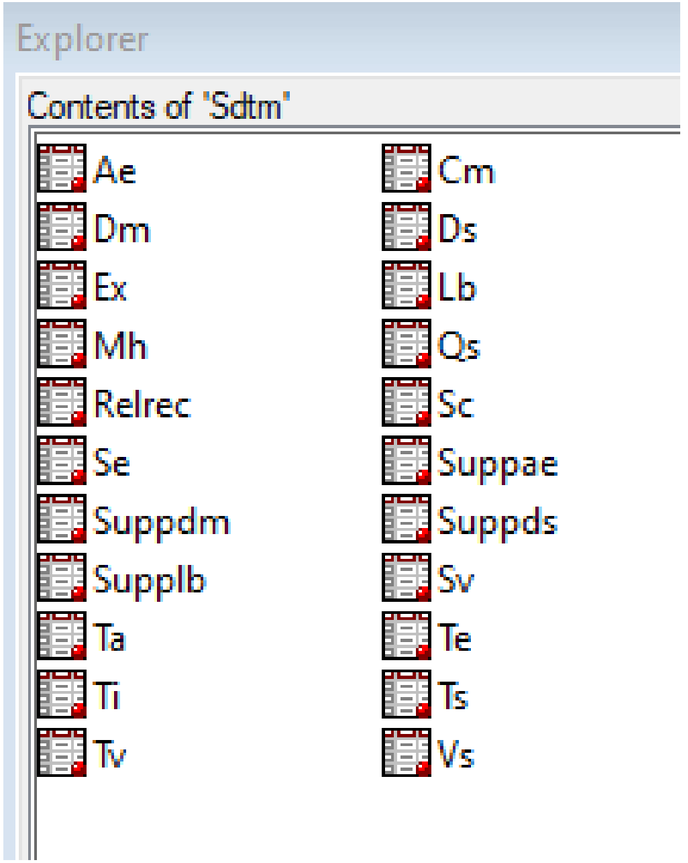
- Dictionary tables provide information about your current SAS session

Dictionary	Contains Information About:
DICTIONARY.TABLES	**Data sets (data set name, number of rows, number of columns, etc.)**
DICTIONARY.COLUMNS	**Variables (variable names, type, length, format, label, etc.)**
DICTIONARY.OPTIONS	**System options (option name, current setting, etc.)**
DICTIONARY.MACROS	**Macro variables (name, value, scope, etc.)**
DICTIONARY.TITLES	**Currently defined titles and footnotes (title number, title text, etc.)**
DICTIONARY.FORMATS	**Currently defined formats (format name, type, default width, etc.)**
…and several others…	

- These can facilitate dynamic programming.
- Dictionaries are accessed using the SQL procedure.

Example #4: Retrieving Variable Names

- Goal: Derive last known date alive for each subject by looking at all date variables in all data sets within a library.
- Business Rule:
  - Date variables have names ending in DTC.
  - Ignore dates in certain data sets that don't contain subject data
- Programming Approach:
  - Stack all date values together in a single data set
  - Invoke PROC SUMMARY to get the last (maximum) date value for each subject.



Example #4: Retrieving Variable Names

Muggle Code – Slide 1 of 3

This DATA step creates a data set with one row for every date value in the entire library.

```
data alldates;
  set
    sdtm.ae(keep=usubjid aedtc    rename=(aedtc   = anydtc))
    sdtm.ae(keep=usubjid aestdtc  rename=(aestdtc = anydtc))
    sdtm.ae(keep=usubjid aeendtc  rename=(aeendtc = anydtc))
    sdtm.cm(keep=usubjid cmdtc    rename=(cmdtc   = anydtc))
    sdtm.cm(keep=usubjid cmstdtc  rename=(cmstdtc = anydtc))
    sdtm.cm(keep=usubjid cmendtc  rename=(cmendtc = anydtc))
    sdtm.dm(keep=usubjid rfstdtc  rename=(rfstdtc = anydtc))
    sdtm.dm(keep=usubjid rfendtc  rename=(rfendtc = anydtc))
    sdtm.dm(keep=usubjid rfxstdtc rename=(rfxstdtc= anydtc))
    sdtm.dm(keep=usubjid rfxendtc rename=(rfxendtc= anydtc))
    sdtm.dm(keep=usubjid rficdtc  rename=(rficdtc = anydtc))
    sdtm.dm(keep=usubjid rfpendtc rename=(rfpendtc= anydtc))
    sdtm.dm(keep=usubjid dthdtc   rename=(dthdtc  = anydtc))
    sdtm.dm(keep=usubjid dmdtc    rename=(dmdtc   = anydtc))
    sdtm.ds(keep=usubjid dsdtc    rename=(dsdtc   = anydtc))
    sdtm.ds(keep=usubjid dsstdtc  rename=(dsstdtc = anydtc))
```

Every date variable is manually coded in a SET statement.

sas innovate

Example #4: Retrieving Variable Names

Muggle Code – Slide 2 of 3

```
    sdtm.ex(keep=usubjid exstdtc  rename=(exstdtc = anydtc))
    sdtm.ex(keep=usubjid exendtc  rename=(exendtc = anydtc))
    sdtm.lb(keep=usubjid lbdtc    rename=(lbdtc   = anydtc))
    sdtm.mh(keep=usubjid mhdtc    rename=(mhdtc   = anydtc))
    sdtm.mh(keep=usubjid mhstdtc  rename=(mhstdtc = anydtc))
    sdtm.qs(keep=usubjid qsdtc    rename=(qsdtc   = anydtc))
    sdtm.sc(keep=usubjid scdtc    rename=(scdtc   = anydtc))
    sdtm.se(keep=usubjid sestdtc  rename=(sestdtc = anydtc))
    sdtm.se(keep=usubjid seendtc  rename=(seendtc = anydtc))
    sdtm.sv(keep=usubjid svstdtc  rename=(svstdtc = anydtc))
    sdtm.sv(keep=usubjid svendtc  rename=(svendtc = anydtc))
    sdtm.vs(keep=usubjid vsdtc    rename=(vsdtc   = anydtc))
  ;
  if length(anydtc)>=10 then do;
    anydt = input(anydtc,e8601da.);
    output;
  end;
  format anydt yymmdd10.;
run;
```

SET statement continued from previous slide

Convert complete dates to numeric date values.

Example #4: Retrieving Variable Names

Muggle Code – Slide 3 of 3

```
proc summary data=alldates nway;
    class usubjid;
    var anydt;
    output
        out=lastdate(drop=_:)
        max(anydt)=lastdt;
run;
```

Get the last date for each subject.

Example #4: Retrieving Variable Names

Macro Wizard Code – Slide 1 of 3

```
proc sql;
   select memname, name
     into :ds1-, :var1-
     from dictionary.columns
    where substr(reverse(strip(name)),1,3) = 'CTD'
      and libname = 'SDTM'
      and memname in (
        select distinct memname from dictionary.columns
        where libname='SDTM' and name = 'USUBJID');
   %let numdates = &sqlobs;
quit;
```

Create two macro variable lists – one for data set names, one for variable names

Only variables with name ending in DTC

Only from data sets also having the variable USUBJID

sas innovate

Example #4: Retrieving Variable Names

Macro Wizard Code – Slide 2 of 3

```
%macro get_all_study_dates;
  data alldates;
    set
     %do i = 1 %to &numdates;
       sdtm.&&ds&i(keep=usubjid &&var&i rename=(&&var&i=anydtc))
     %end;
    ;
    if length(anydtc)>=10 then do;
      anydt = input(anydtc,e8601da.);
      output;
    end;
    format anydt yymmdd10.;
  run;
%mend;
%get_all_study_dates;
```

Build the SET statement dynamically by looping through the macro variable lists.

Remainder of DATA step is the same as before

Example #4: Retrieving Variable Names

Macro Wizard Code – Slide 3 of 3

```
proc summary data=alldates nway;
    class usubjid;
    var anydt;
    output
        out=lastdate(drop=_:)
        max(anydt)=lastdt;
run;
```

PROC SUMMARY is the same as before

Applying Macro Magic to Environmental Data

Spell #3

sas innovate

Incantation #4: The %SYSFUNC Function

- The macro language has a limited set of functions (a couple dozen)
- The DATA step has an extensive library of functions (over 400)
- The %SYSFUNC bridging function allows the use of the vast majority of DATA step functions within the macro language

```
%sysfunc(datastepfunction(args)<,format>)
```

- Examples:

```
%let currdate = %sysfunc(date(),yymmdd10.);

%if %sysfunc(exist(work.mydata)) %then %do ...
```

Example #5: OS-specific Execution

- SAS practitioners may work in different environments.
- Muggle approach: Separate programs for each environment
  - The programmer needs to maintain multiple sets of programs.
- Macro Wizard approach: Use automatic macro variables
  - The code is set up so that it can still execute regardless of the environment in which it is run.

Example #5: OS-specific Execution

```
%macro envchk;
   %if &sysscp = WIN %then %do;
      %let ppcmd = %str(dir);
   %end;
   %else %if &sysscp = LIN X64 %then %do;
      %let ppcmd = %str(ls -l);
   %end;
   %else %do;
      %put %sysfunc(compress(E RROR:)) ENVIRONMENT NOT SPECIFIED;
      %abort;
   %end;
   %put &=ppcmd;
   /* additional SAS code */
%mend envchk;
%envchk
```

&SYSSCP indicates the environment in which SAS is being run.

Successful in identifying environment

```
PPCMD=dir
```

Not successful in identifying environment

```
ERROR: ENVIRONMENT NOT SPECIFIED
ERROR: Execution terminated by an %ABORT statement.
```

Example #6: Date-Specific Execution

- Portions of a program may need to be executed on specific days of the month or days of the week.
- Muggle approach: Separate programs for each combination of days of the month and days of the week
  - The programmer needs to maintain multiple sets of programs.
- Macro Wizard approach: Use automatic macro variables
  - The code is set up so that it will always execute the regularly scheduled portion but only execute the portions that are day specific when necessary.

Example #6: Date-Specific Execution

- System macro variables can also be used to run code at specific date, day or time

```
%if &sysday = Monday %then %do;
   /*** SAS Code that only runs on Monday ***/
%end;

%if %sysfunc(day("&sysdate"d)) = 1 %then %do;
   /*** SAS Code that only runs on first of month ***/
%end;
```

Represents the date SAS session started

- System macro variables can also be used to run code at specific date, day or time

```
%sysfunc(date())
```

sas innovate

Example #7a: User-Specific Execution

- Some programmers may not have access to certain data, and we need to control who executes the code.
- Muggle approach: Program bombs when run by user without access to data
- Macro Wizard approach: Use automatic macro variables

Example #7a: User-Specific Execution

&SYSUSERID contains the ID of the user running the program

```
%macro ctrlexec;
    %if &sysuserid = gonza %then %do;
      %let msg = &sysuserid HAS PERMISSION TO EXECUTE;
    %end;
    %else %do;
      %put %sysfunc(compress(E RROR:)) &sysuserid DOES NOT HAVE
PERMISSION TO EXECUTE;
      %abort;
    %end;
    /* additional SAS code */
    %put &=msg;
%mend ctrlexec;

%ctrlexec
```

Allowed in the Restricted Section

```
MSG=gonza HAS PERMISSION TO EXECUTE
```

Not allowed in the Restricted Section

```
ERROR: jhorst DOES NOT HAVE PERMISSION TO EXECUTE
ERROR: Execution terminated by an %ABORT statement.
```

sas innovate

Example #7b: User-Specific Execution

- Some companies may use cloud storage and access is based on a user ID.
- Muggle approach: manually enter the user ID each time a path is specified

```
libname ads
"P:\Users\rwatson\Box\Biometrics\StatProg\Compound\Analysis\Data";
```

- Macro Wizard approach: use automatic macro variables

```
libname ads
"P:\Users\&sysuserid.\Box\Biometrics\StatProg\Compound\Analysis
\Data";
```

sas innovate

Wrap Up

Conclusion

- The SAS Macro Language provides powerful data-driven magic!
- Cast these spells to build robust programs:
  - Include dynamic logic
  - Avoid hard-coding
  - Adapt to changes in data or computing environment
- Advantages:
  - Less likely to require change
  - Easier to maintain
  - Greater potential for reuse

Any Questions?

josh@nestedloopconsulting.com
richann.watson@datarichconsulting.com

sas innovate

Recommended Resources

- Carpenter, Art. 2016. *Carpenter's Complete Guide to the SAS® Macro Language, Third Edition*. Cary, NC: SAS Institute Inc.
- SAS Institute Inc. 2016. *SAS® 9.4 Macro Language: Reference, Fifth Edition*. Cary, NC: SAS Institute Inc.