SAS Macro Efficiencies

Charu Shankar
SAS Institute

Pharmasug 2021 Hands-On Training
Charu Shankar teaches by engaging with logic, visuals and analogies to spark critical thinking. She interviews users to recommend the right SAS training.

**SAS blogger**, yoga teacher & **chef**, Charu also helps support users looking to land work using SAS through [Linkedin](https://www.linkedin.com).

In addition to teaching public classes, and academia, Charu has presented at over 150 SAS international conferences on SAS, SAS Viya, SQL, Macro, Hadoop, Python, DS2, tips and tricks, efficiencies.
1. Intro- Why Macro
2. Program Flow
2. Create Macro Variables For Text Substitution
3. Create Macro Programs To Generate Code
4. Store A query In A Macro To Reuse
5. Handy Links
I. Why Macro
The SAS macro facility enables you to write code that rewrites itself!
Why Macro
1. Substituting User-Defined Data Values

```sas
title "Trucks by Origin";
proc freq data=sashelp.cars;
  where Type="Truck";
  table Origin;
run;

title "Average Highway MPG for Trucks";
proc means data=sashelp.cars mean maxdec=1;
  where Type="Truck";
  var MPG_Highway;
  class Origin;
run;
```

Easily replace repetitive values.
Why Macro
2. Conditional Processing

data mpg;
    set sashelp.cars;
    AvgMPG=mean(MPG_Highway, MPG_City);
run;

Did the step run without warnings or errors?

YES
 Print the table.

NO
 Write a custom error message to the log.

ERROR: MPG table was not created successfully.

Submit or modify code based on a condition.

proc print data=mpg;
    run;

CarsProgram.sas
Why Macro
3. Repetitive Processing

```sas
title "4-Cylinder Cars";
proc print data=sashelp.cars;
  where Cylinders=4;
run;

title "6-Cylinder Cars";
proc print data=sashelp.cars;
  where Cylinders=6;
run;

title "8-Cylinder Cars";
proc print data=sashelp.cars;
  where Cylinders=8;
run;
```

Generate repetitive SAS code - based on different data values (data driven processing).

CarsProgram.sas
Why Macro
4. Data-Driven Applications

```sas
data hybrid sedan sports suv truck wagon;
set sashelp.cars;
select (Type);
  when("Hybrid") output hybrid;
  when("Sedan") output sedan;
  when("Sports") output sports;
  when("SUV") output suv;
  when("Truck") output truck;
  when("Wagon") output wagon;
  otherwise;
end;
run;
```

Build programs dynamically based on data values (what if data set is upgraded to include LUXURY models).

CarsProgram.sas
Why Macro
5. Substituting System Values

Automatically substitute system values into a program.

How can the macro language make your job easier as a SAS programmer?

```sas
title1 "Cars List";
title2 "Created at 10:24 AM on October 9, 2019";
footnote "Environment: SAS 9.4 on Win X64_10PRO";

proc print data=sashelp.cars;
run;
```
Why Macro 6. Flexibility

```sas
title "    s with Horsepower >    ";
proc print data=sashelp.cars;
    var Make Model MSRP Horsepower;
    where Type=":    " and Horsepower>"    ";
run;
```

Macro variables store text that can be used anywhere in our SAS programs.

Truck  250
Sedan  150
SUV    200
Why Macro Placeholder
7. Store SQL Query Values

```
title "4-Cylinder Cars";
proc print data=sashelp.cars;
  where Cylinders=4;
run;

title "6-Cylinder Cars";
proc print data=sashelp.cars;
  where Cylinders=6;
run;

title "8-Cylinder Cars";
proc print data=sashelp.cars;
  where Cylinders=8;
run;
```
Efficiency of Macro-Based Applications

The macro facility processes the text in a program to automate and customize the code.

#PharmaSUG2021
The macro language won’t make your code run faster, but it can reduce your development and maintenance time.
1.1 Seeing the Macro Language in Action
1.01 Activity

Open **CarsMacro.sas** run it segment by segment to get answers to all the questions posed before in this section.
II. Program Flow
SAS Programming Languages

- **DATA step language**: data manipulation
- **PROC SQL**: data manipulation and reporting
- **SAS procedures**: data analysis and reporting

SAS program → Compile → Execute
SAS Macro Language

- DATA step language: data manipulation
- PROC SQL: data manipulation and reporting
- SAS procedures: data analysis and reporting
- SAS macro language: generate SAS program code

Text → Macro Facility → SAS program

%f(x)
Conceptual SAS Program Flow

- SAS code
- Compiler
- Macro code
- Macro Processor
title "Trucks with Horsepower > 250";
proc print data=sashelp.cars;
   var Make Model MSRP Horsepower;
   where Type="Truck" and Horsepower>250;
run;
A program is separated into components called tokens. There are 4 types of tokens.

1. **name**
2. **number**
3. **special**
4. **literal**

```plaintext
title "Trucks with Horsepower > 250";
proc print data=sashelp.cars;
   var Make Model MSRP Horsepower;
   where Type="Truck" and Horsepower>250;
run;
```
Name Tokens

Input Stack

```plaintext
title "Trucks with Horsepower > 250";
proc print data=sashelp.cars;
    var Make Model MSRP Horsepower;
    where Type="Truck" and Horsepower > 250;
run;
```

- maximum of 32 characters
- must begin with a letter or underscore
- can include only letters, digits, and underscores
Number Tokens

Input Stack

title "Trucks with Horsepower > 250";
proc print data=sashelp.cars;
  var Make Model MSRP Horsepower;
  where Type="Truck" and Horsepower>250;
run;

• integer or floating-point values
• digits, decimal point, leading sign, and exponent indicator (e or E)
• date/time/datetime constants:
  '29APR2019'd
  '14:05:32.1't
  '29APR2019 14:05:32.1'dt
Literal Tokens

Input Stack

```sas
title "Trucks with Horsepower > 250";
proc print data=sashelp.cars;
  var Make Model MSRP Horsepower;
  where Type="Truck" and Horsepower>250;
run;
```

- string of any characters enclosed in single or double quotation marks
- maximum of 32,767 characters
- treated as a single unit

literal
Special Tokens

Input Stack

```sas
title "Trucks with Horsepower > 250";
proc print data=sashelp.cars;
    var Make Model MSRP Horsepower;
    where Type="Truck" and Horsepower>250;
run;
```

- any character or group of characters that have special meaning to SAS
- examples:
  * / + - ; ( ) . & %

special
Token Delimiters

Input Stack

```
title "Trucks with Horsepower > 250";
proc print data=sashelp.cars;
  var Make Model MSRP Horsepower;
  where Type="Truck" and Horsepower>250;
run;
```

Blanks separate tokens.

A token ends when a new type of token begins.
2.01 Activity

How many tokens are in each statement?

```plaintext
title "Trucks with Horsepower > 250";
proc print data=sashelp.cars;
  var Make Model MSRP Horsepower;
  where Type="Truck" and Horsepower>250;
run;
```
How many tokens are in each statement?

title "Trucks with Horsepower > 250";
proc print data=sashelp.cars;
    var Make Model MSRP Horsepower;
    where Type="Truck" and Horsepower>250;
run;
Why do I need to know about tokenization?

It can help you understand the timing of how SAS and macro language elements are processed.

title "Trucks with Horsepower > 250";
proc print data=sashelp.cars;
  var Make Model MSRP Horsepower;
  where Type="Truck" and Horsepower>250;
run;
The word scanner reads individual tokens and passes them on to the compiler.

```
"Trucks with Horsepower > 250";
proc print data=sashelp.cars;
  var Make Model MSRP Horsepower;
  where Type="Truck" and Horsepower>250;
run;
```
### Tokenization

**Compiler**

```plaintext
title "Trucks with Horsepower > 250"
```

**Word Scanner**

<table>
<thead>
<tr>
<th>literal start</th>
<th>&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Trucks</td>
</tr>
<tr>
<td>name</td>
<td>with</td>
</tr>
<tr>
<td>name</td>
<td>Horsepower</td>
</tr>
<tr>
<td>special</td>
<td>&gt;</td>
</tr>
<tr>
<td>number</td>
<td>250</td>
</tr>
<tr>
<td>literal end</td>
<td>&quot;</td>
</tr>
</tbody>
</table>

**Input Stack**

```
proc print data=sashelp.cars;
  var Make Model MSRP Horsepower;
  where Type="Truck" and Horsepower>250;
run;
```
title "Trucks with Horsepower > 250";

special ;

proc print data=sashelp.cars;
    var Make Model MSRP Horsepower;
    where Type="Truck" and Horsepower>250;
run;
```plaintext
name = special
data
print
cars;
var Make Model MSRP Horsepower;
where Type="Truck" and Horsepower>250;
run;
```
```
proc print data=sashelp.cars;
  MSRP  name  Model  name  var
  where Type="Truck" and Horsepower>250;
run;
```
```
proc print data=sashelp.cars;
  var Make Model MSRP Horsepower;
  where Type="Truck" and Horsepower>250;
run;
```
Macro Triggers

The combination of a % or & special token immediately followed by a name token is called a *macro trigger*.
Macro Triggers

%PUT text;

- %PUT is a macro statement that writes text to the log.
- Text is not quoted.
- Use NOTE:, WARNING:, or ERROR: prefix to color the log message.

%put NOTE: This is a great program!;

73 %put NOTE: This is a great program!;
NOTE: This is a great program!
Macro Triggers

%PUT text;

- %PUT is a macro statement that writes text to the log.
- Text is not quoted.
- Use NOTE:, WARNING:, or ERROR: prefix to color the log message.

71 %put NOTE: This Is A Great Program!;
NOTE: This Is A Great Program!

72 %put WARNING: You May Want To Check Out The Warning!;
WARNING: You May Want To Check Out The Warning!

73 %put ERROR: This program has errors!;
ERROR: This program has errors!
Macro triggers modify word scanner behavior.

Compiler

Word Scanner

Input Stack

%put NOTE: This is a great program!;
Tokenizing Code with Macro Triggers

NOTE: This is a great program!;
Tokenizing Code with Macro Triggers

**Macro Processor**

```plaintext
%put NOTE: This is a great program!
```

**Word Scanner**

**Compiler**

**Input Stack**

#PharmaSUG2021
%put NOTE: This is a great program!
title "Trucks with Horsepower > 250";
proc print data=sashelp.cars;
var Make Model MSRP Horsepower;
where Type="Truck" and Horsepower>250;
run;
## 3. Create Macro Variables For Text Substitution

<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Creating Macro Variables</td>
</tr>
<tr>
<td>3.2 Resolving Macro Variables</td>
</tr>
<tr>
<td>3.3 Troubleshooting</td>
</tr>
</tbody>
</table>
3.1 Creating Macro Variables with `%LET`
3.1 Creating Macro Variables with %LET

Macro variable names:
• follow SAS naming rules
• are stored as uppercase
• are not case sensitive

%LET name=value;

%let type=Truck;
%let hp=250;

Global Symbol Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>Truck</td>
</tr>
<tr>
<td>HP</td>
<td>250</td>
</tr>
</tbody>
</table>
3.1 Creating Macro Variables with %LET

%LET name=value;

%let type=Truck;
%let hp=250;

- Case is preserved.
- Leading and trailing blanks are removed.
- It stores 0 to 65,534 (64K) characters.
- The length is dynamically set each time a value is assigned.

Global Symbol Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>Truck</td>
</tr>
<tr>
<td>HP</td>
<td>250</td>
</tr>
</tbody>
</table>
3.1 Creating Macro Variables with %LET

%let type=Truck;
%let hp=250;
%let type= Sports ;
%let origin=" Europe ";
%let value=;
%let sum=3+4;
%let varlist=Make Model Type;

### Global Symbol Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>Truck</td>
</tr>
<tr>
<td>HP</td>
<td>250</td>
</tr>
</tbody>
</table>

Macro variables don't have a type of character or numeric. All values are stored as text.
Leading and trailing spaces are removed. The value of an existing macro variable is replaced.
3.1 Creating Macro Variables with `%LET`

Quotation marks are stored as part of the value.

```sas
%let type=Truck;
%let hp=250;
%let type= Sports ;
%let origin=" Europe ";
%let value=;
%let sum=3+4;
%let varlist=Make Model Type;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>Sports</td>
</tr>
<tr>
<td>HP</td>
<td>250</td>
</tr>
<tr>
<td>ORIGIN</td>
<td>&quot; Europe &quot;</td>
</tr>
</tbody>
</table>

Global Symbol Table
3.1 Creating Macro Variables with %LET

```
%let type=Truck;
%let hp=250;
%let type=Sports;
%let origin="Europe";
%let value=;
%let sum=3+4;
%let varlist=Make Model Type;
```

Global Symbol Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>Sports</td>
</tr>
<tr>
<td>HP</td>
<td>250</td>
</tr>
<tr>
<td>ORIGIN</td>
<td>&quot;Europe&quot;</td>
</tr>
<tr>
<td>VALUE</td>
<td></td>
</tr>
</tbody>
</table>

A null value is stored.
3.1 Creating Macro Variables with %LET

%let type=Truck;
%let hp=250;
%let type=Sports;
%let origin="Europe";
%let value=;
%let sum=3+4;
%let varlist=Make Model Type;

Global Symbol Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>Sports</td>
</tr>
<tr>
<td>HP</td>
<td>250</td>
</tr>
<tr>
<td>ORIGIN</td>
<td>&quot;Europe&quot;</td>
</tr>
<tr>
<td>VALUE</td>
<td></td>
</tr>
<tr>
<td>SUM</td>
<td>3+4</td>
</tr>
</tbody>
</table>

Mathematical expressions are not evaluated.
3.1 Creating Macro Variables with %LET

The variable list is stored as a text string.

Global Symbol Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>Sports</td>
</tr>
<tr>
<td>HP</td>
<td>250</td>
</tr>
<tr>
<td>ORIGIN</td>
<td>&quot; Europe &quot;</td>
</tr>
<tr>
<td>VALUE</td>
<td></td>
</tr>
<tr>
<td>SUM</td>
<td>3 + 4</td>
</tr>
<tr>
<td>VARLIST</td>
<td>Make Model Type</td>
</tr>
</tbody>
</table>
3.1 Creating Macro Variables with %LET

3.01 Activity

What would be stored as the value of Mylib?

%let mylib=libname mcl "s:/workshop";

<table>
<thead>
<tr>
<th>Global Symbol Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>MYLIB</td>
</tr>
</tbody>
</table>
What would be stored as the value of Mylib?

```
%let mylib=libname mc1 "s:/workshop";
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MYLIB</td>
<td>libname mc1 &quot;s:/workshop&quot;</td>
</tr>
</tbody>
</table>

The semicolon is treated as the conclusion of the %LET statement and is not stored in the macro variable value.
Macro variables remain in the global symbol table for the duration of the SAS session.

### Global Symbol Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>Sports</td>
</tr>
<tr>
<td>HP</td>
<td>250</td>
</tr>
<tr>
<td>ORIGIN</td>
<td>&quot; EUROPE &quot;</td>
</tr>
<tr>
<td>VALUE</td>
<td></td>
</tr>
<tr>
<td>SUM</td>
<td>3+4</td>
</tr>
<tr>
<td>VARLIST</td>
<td>Make Model Type</td>
</tr>
</tbody>
</table>
3.1 Creating Macro Variables with %LET

When the SAS session ends, the global symbol table is deleted.

Global Symbol Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORIGIN</td>
<td>&quot; EUROPE &quot;</td>
</tr>
<tr>
<td>VALUE</td>
<td></td>
</tr>
<tr>
<td>SUM</td>
<td>3+4</td>
</tr>
<tr>
<td>VARLIST</td>
<td>Make Model Type</td>
</tr>
</tbody>
</table>
3.2 Resolving Macro Variables

substitutes the macro variable value into the program

&name

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>Truck</td>
</tr>
<tr>
<td>HP</td>
<td>250</td>
</tr>
</tbody>
</table>

%let type=Truck;
%let hp=250;

proc print data=sashelp.cars;
  var Make Model MSRP Horsepower;
  where Type="&type" and Horsepower>&hp;
run;
What must be modified in the program to generate a list of SUVs with horsepower greater than 300, and then print the date in the footnote?
3.2 Resolving Macro Variables
Updating Macro Variables

Simply update the %LET statements!
3.3 Troubleshooting Macro Variable References

```
OPTIONS SYMBOLGEN | NOSYMBOLGEN;
options symbolgen;
%let type=Truck;
%let hp=250;
title1 "Car Type: &type";
proc print data=sashelp.cars;
   var Make Model MSRP Horsepower;
   where Type="&type" and Horsepower>&hp;
run;
```

The SYMBOLGEN option writes information to the log when macro variable references resolve.

```
80 where Type="&type" and Horsepower>&hp;
SYMBOLGEN: Macro variable TYPE resolves to Truck
SYMBOLGEN: Macro variable HP resolves to 250
```
Open **m103p02.sas** from the **activities** folder and perform the following tasks:

1. Notice that the program includes two TITLE statements, each referencing a macro variable.
2. At the top of the program, turn on the SYMBOLGEN option. At the bottom of the program, turn off SYMBOLGEN.

   ```
   options symbolgen;
   options nosymbolgen;
   ```

3. Run the program and review the log and results. What is printed as the second title?
4. In the TITLE2 statement, change the single quotation marks to double quotation marks and run the program again. How do the results and the log differ?
3.3 Troubleshooting Macro Variable References

3.02 Activity – Correct Answer

3. What is printed as the second title?

 SYMBOLGEN: Macro variable TYPE resolves to Truck
 76 title1 "Car Type: &type";  
 77 title2 'Horsepower > &hp';

The macro variable does not resolve with single quotation marks.

4. How do the results and the log differ?

 SYMBOLGEN: Macro variable TYPE resolves to Truck
 76 title1 "Car Type: &type";
 SYMBOLGEN: Macro variable HP resolves to 250
 77 title2 "Horsepower > &hp";

The macro variable resolves with double quotation marks.
3.3 Troubleshooting Macro Variable References

Quotation Marks in Literal Tokens

```
title1 "Car Type: &type";
title2 'Car&Power Report';
```

Macro triggers in double quotation marks are sent to the macro processor.

Macro triggers in single quotation marks are treated as regular text and are not resolved.
3.3 Troubleshooting Macro Variable References

Delimiting Macro Variable References

...%let type=Truck;
title "&type\text{ with Horsepower} > &hp";
...

What happens if a macro variable reference is concatenated with trailing text?

<table>
<thead>
<tr>
<th>Obs</th>
<th>Make</th>
<th>Model</th>
<th>MSRP</th>
<th>Horsepower</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Cadillac</td>
<td>Escalade EXT</td>
<td>$52,975</td>
<td>345</td>
</tr>
<tr>
<td>85</td>
<td>Chevrolet</td>
<td>Avalanche 1500</td>
<td>$36,100</td>
<td>295</td>
</tr>
<tr>
<td>88</td>
<td>Chevrolet</td>
<td>Silverado SS</td>
<td>$40,340</td>
<td>300</td>
</tr>
<tr>
<td>88</td>
<td>Chevrolet</td>
<td>Silverado SS</td>
<td>$41,995</td>
<td>300</td>
</tr>
</tbody>
</table>
3.3 Troubleshooting Macro Variable References

Delimiting Macro Variable References

```r
...  
%let type=Truck;  
title "&types with Horsepower > &hp";  
...
```

Global Symbol Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>Truck</td>
</tr>
<tr>
<td>HP</td>
<td>250</td>
</tr>
</tbody>
</table>

TYPES is not found.

74 %let type=Truck;
75 %let hp=250;
WARNING: Apparent symbolic reference TYPES not resolved.
SYMBOLGEN: Macro variable HP resolves to 250
76 title "&types with Horsepower > &hp";
3.3 Troubleshooting Macro Variable References

Delimiting Macro Variable References

...%let type=Truck;
title "&type.s with Horsepower > &hp";
...

Use a period to delimit the macro variable name from the text.

title "Trucks with Horsepower > 250";

The period does not appear in the resolved text.
3.3 Troubleshooting Macro Variable References

3.03 Activity

Open **m103p03.sas** from the **activities** folder and perform the following tasks:

1. Modify the TITLE statement reference to **&type** by adding a period before the 's'.

2. Replace the hardcoded text **sashelp** in the FOOTNOTE and PROC statements with a reference to the **Lib** macro variable (**&lib**).

   ```sas
   footnote "Data Source: &lib.CARS";
   proc print data=&lib.cars;
   ```

3. Run the program and examine the log and the error statements. Why did the program fail to run?
3. Run the program and examine the log and the error statements. Why did the program fail to run?

The period between &lib and cars was interpreted as a macro delimiter, so the table name incorrectly resolved to SASHELPCars.

```
80 proc print data=&lib.cars;
SYMBOLGEN: Macro variable LIB resolves to SASHELP
ERROR: File WORK.SASHELPCars.DATA does not exist.
```
### 3.3 Troubleshooting Macro Variable References

**Delimiting Macro Variable References**

footnote "Data Source: &lib...CARS";
proc print data=&lib..cars;

Use two periods between the macro variable and table name.

The first period is a delimiter and is removed when &lib resolves. The second period remains as text.
3.3 Troubleshooting Macro Variable References
Displaying Macro Variables

```bash
%PUT &=name;

73 %put &=type;
    TYPE=Truck

%PUT <text> &name <text>;

75 %put The value of Horsepower is &hp;
    The value of Horsepower is 250
```
1. Open a new program and submit a %PUT statement to list all user-defined macro variables.

2. Find the **Path** macro variable in the log. **Path** was created with a %LET statement in the *libname.sas* program, and it stores the location of the course files. Submit the following statements to view the value of **Path**. How are the messages in the log different?

```sas
%put NOTE: &=path;
%put ERROR- Course files are in &path;
```
3.3 Troubleshooting Macro Variable References

3.04 Activity – Correct Answer

How are the messages in the log different?

The colon includes the word **NOTE** and prints the message in the default color for notes.

72 %put NOTE: &path;
NOTE: PATH=/home/u47296115/my_shared_file_links/u47296115/Shankar

73 %put ERROR- Course files are in &path;
Course files are in /home/u47296115/my_shared_file_links/u47296115/Shankar

The hyphen changes the color of the text but does not print the prefix.

The path for your course files might differ.
Several useful automatic macro variables are defined when you start a SAS session.

```sas
%PUT _AUTOMATIC_;

73 %put _automatic_;
...
AUTOMATIC SYSDATE 01NOV19
AUTOMATIC SYSDATE9 01NOV2019
AUTOMATIC SYSDAY Friday
AUTOMATIC SYSENCODING wlatin1
AUTOMATIC SYSUSERID msmith
AUTOMATIC SYSVER 9.4
...
```

lists all automatic macro variables created by SAS
3.3 Troubleshooting Macro Variable References

3.05 Activity

Open m103p04.sas from the activities folder and perform the following tasks:

1. Review the program and notice that the DATA step creates a table named **Avg_MPG**. Highlight the DATA step and %PUT statement and run the selected code. Review the log to see all automatic macro variables stored in the global symbol table.

2. Identify the macro variables that store the date and the last table created.

3. Use macro variable references in the TITLE2 and FOOTNOTE statements to insert the table name and date into the program.

```sas
title2 "Data Source: <table>";
footnote "Created on <date>";
```
3.3 Troubleshooting Macro Variable References

3.05 Activity – Correct Answer

3. Use macro variable references in the TITLE2 and FOOTNOTE statements to insert the table name and date into the program.

```
title2 "Data Source: &syslast";
footnote "Created on &sysdate9";
```
### 3.3 Troubleshooting Macro Variable References

#### Selected Automatic Macro Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDAY</td>
<td>Day of the week on which the SAS session was initiated</td>
</tr>
<tr>
<td>SYSDATE9</td>
<td>Date on which the SAS session was initiated</td>
</tr>
<tr>
<td>SYSTIME</td>
<td>Time at which the SAS session was initiated</td>
</tr>
<tr>
<td>SYHOSTNAME</td>
<td>Name of the machine on which the SAS session is executing</td>
</tr>
<tr>
<td>SYSLAST</td>
<td>Name of the last table created</td>
</tr>
<tr>
<td>SYSUSERID</td>
<td>User ID under which the SAS session is executing</td>
</tr>
<tr>
<td>SYSSCP</td>
<td>Operating system abbreviated name</td>
</tr>
<tr>
<td>SYSSCPL</td>
<td>Operating system detail</td>
</tr>
<tr>
<td>SYSVLONG</td>
<td>SAS version and maintenance release</td>
</tr>
<tr>
<td>SYSERR</td>
<td>Returns a 0 if a step completes successfully and without warning messages, returns an error code otherwise</td>
</tr>
</tbody>
</table>
%PUT _USER_;  

lists all macro variables created by the user or application (SAS Studio or SAS Enterprise Guide)

74 %put _user_;  
...  
GLOBAL CLIENTMACHINE L12345  
GLOBAL HP 250  
GLOBAL LIB sashelp  
GLOBAL PATH s:/workshop  
GLOBAL SYSUSERNAME John Smith  
GLOBAL TYPE Truck  
GLOBAL _CLIENTAPP 'SAS Studio'  
GLOBAL _CLIENTAPPVERSION 3.8  
...

%PUT is a simple way to view macro variable names and values in the log.
IV. Create Macro Programs To Generate Code
IV. Create Macro Programs To Generate Code

4.1 Defining and Calling a Macro

4.2 Conditional Processing

4.3 Iterative Processing
4.1 Defining and Calling a Macro

Developing Macro Applications

- Start with a validated SAS program
- Generalize with macro variables
- Create a macro definition with parameters
- Use macro-level programming for complex processing
- Add Data Driven Features
- Validate parameters and document

```sas
proc print data=sashelp.cars(obs=5);
run;
```
4.1 Defining and Calling a Macro

Developing Macro Applications

- Start with a validated SAS program
- Generalize with macro variables
- Create a macro definition with parameters
- Use macro-level programming for complex processing
- Add Data Driven Features
- Validate parameters and document

%let dsn=sashelp.cars;
%let obs=5;
proc print data=&dsn(obs=&obs);
run;
%let dsn=sashelp.cars;
%let obs=5;
proc print data=&dsn(obs=&obs);
run;

How can I turn this program into a reusable macro definition?
4.1 Defining and Calling a Macro

Macro Definitions

```sas
%MACRO macro-name </DES="description">;
  macro-text
%MEND <macro-name>;

%let dsn=sashelp.cars;
%let obs=5;
%macro printable / des="Print a table";
proc print data=&dsn(obs=&obs);
  run;
%mend printable;
```

`macro-text` can include:
- macro language statements or expressions
- complete or partial SAS program statements or program steps.
4.1 Defining and Calling a Macro

Macro Definitions

```sas
%macro printable / des="Print a table";
proc print data=&dsn(obs=&obs);
run;
%mend printable;
```

Compiled *macro definitions* are stored in catalogs in a SAS library.
4.1 Defining and Calling a Macro

4.01 Activity

Open m104p02.sas from the activities folder and perform the following tasks:

1. Notice that there is a missing semicolon after the PROC PRINT statement. Run the program and confirm that an error is generated in the log. Do not correct the error.

2. Modify the code to create a macro program named PrintTable. Run the program and notice that there are no notes, warnings, or errors in the log.

```sas
%macro printtable;
proc print data=&dsn(obs=&obs) run;
%mend printtable;
```
3. Add the following OPTIONS statements at the start of the program to set the MCOMPILENOTE=ALL option.

```plaintext
options mcompilenote=all;
```

4. Submit the program. The log includes a note indicating that the `PrintTable` macro compiled without errors.

5. Why does the macro program compile successfully even though there is a syntax error in the PROC PRINT step?
Why does the macro program compile successfully even though there is a syntax error in the PROC PRINT step?

73 options mcompilenote=all;
74
75 %let dsn=sashelp.cars;
76 %let obs=5;
77
78 %macro printtable;
79 proc print data=&dsn(obs=&obs)
80 run;
81 %mend printtable;

NOTE: The macro PRINTTABLE completed compilation without errors.
4.1 Defining and Calling a Macro

Macro Definitions

```sas
OPTIONS MCOMPILENOTE=ALL|NONE;

%let dsn=sashelp.cars;
%let obs=5;
options mcompilenote=all;

%macro printable;
proc print data=&dsn(obs=&obs);
run;
%mend printable;

options mcompilenote=none;
```

Use MCOMPILENOTE=ALL to confirm that your macro definition compiled successfully.

**NOTE:** The macro PRINTTABLE completed compilation without errors.
### 4.1 Defining and Calling a Macro

#### Calling a Macro

A macro call executes the code stored in the macro definition.

```sas
%macro macro-name

%let dsn=sashelp.cars;
%let obs=5;

%printtable

Do not include a semicolon after the macro call.
```

---

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Type</th>
<th>Create Date</th>
<th>Modified Date</th>
<th>Description</th>
</tr>
</thead>
</table>
4.1 Defining and Calling a Macro

4.02 Activity

Open m104p03.sas from the activities folder and perform the following tasks:

1. Run the program and verify in the log that the %PrintTable macro compiled successfully.

2. Add a line of code at the bottom of the program to call the %PrintTable macro.

3. Highlight the macro call and run the selected code. Based on only what you see in the log, can you tell where in the SAS program the error occurred?
Based on only what you see in the log, can you tell where in the SAS program the error occurred?

```sas
%printtable
ERROR: File SASHELP.CLAS.DATA does not exist.
NOTE: The SAS System stopped processing this step because of errors.
```

The error is displayed in the log, but the code is not displayed.
4.1 Defining and Calling a Macro

Troubleshooting a Macro

```sas
options mprint;
%printtable
options nomprint;

84 %printtable
MPRINT(PRINTTABLE): proc print data=sashelp.clas(obs=5);
ERROR: File SASHELP.CLAS.DATA does not exist.
MPRINT(PRINTTABLE): run;

name of the macro generating the code
SAS code generated by the macro
```

The displayed code has all macro triggers already resolved.
%macro printtable;
proc print data=&dsn(obs=&obs);
run;
%mend printtable;

%let dsn=sashelp.cars;
%let obs=5;
%printtable

In order for the %PrintTable macro to run successfully, values must first be assigned to the included macro variables.
Using macro parameters is a practical alternative to assigning macro variable values using %LET.

Create a macro definition with parameters

```sas
%macro printtable(dsn,obs);
proc print data=&dsn(obs=&obs);
run;
%mend printtable;

%printtable(sashelp.cars,5)
```
4.1 Defining and Calling a Macro

Macro Positional Parameters

```sas
%MACRO macro-name(parameter-1, ... parameter-n);
%macro-name(value-1, ... value-n)

%macro printable(dsn,obs);
%
printtable(sashelp.cars,5)
```

- In a macro definition, positional parameter names are provided in an ordered list.
- In a macro call, positional parameter values must be listed in the same order.
Creating a Macro Definition with Parameters

This demonstration illustrates the process for creating a macro definition with parameters.
Creating a Macro Definition with Parameters

Scenario

We have a SAS program that produces a report about storms in the 2016 season in the North Atlantic basin with maximum winds greater than 80 MPH. We want to be able to produce the same report for other seasons, basins, and wind speeds.

m104p04.sas

Syntax

\%macro-name(parameter1, ... parameterN)

Notes

- Macro definitions can be created to accept parameters.
- Macro parameters are resolved as macro variables during macro execution.
- Parameters are named in a comma-separated list in parentheses in the %MACRO statement.
- When the macro is called, parameter values are supplied in parentheses as a comma-delimited list.
- Because these parameters are positional, parameter values must be supplied in the proper order.

Demo

1. Open m104p04.sas from the demos folder. This program produces a bar chart to examine the frequency of StormType by Basin. It includes storms in the NA basin (North Atlantic) during the 2016 season with maximum wind speed greater than 80 MPH. Run the program and confirm the results.

![Bar chart showing storm frequency by type](image)

2. Add %LET statements to create macro variables named Basin, Season, and MaxWind, and assign values of NA, 2016, and 80 respectively. Replace hardcoded values in the SAS program with the appropriate macro variable references. Run the program and verify that it reproduces the original report with no errors or warnings in the log.
%let Basin=NA;
%let Season=2016;
%let MaxWind=80;

**title1** "Storm Frequency by Type";
**title2** "&Basin Basin, &Season Season, Max Wind > &MaxWind MPH";
proc sgplot data=mc1.storm_final;
  vbar StormType / dataskin=pressed;
  where Basin="&Basin" and Season=&Season and MaxWindMPH>&MaxWind;
run;
**title**;

3. Test the generalized program by modifying the %LET statements for **Basin**, **Season**, and **MaxWind** to assign values of EP, 2015, and 125 respectively. Run the program and verify that it runs without producing errors or warnings in the log.

%let Basin=EP;
%let Season=2015;
%let MaxWind=125;

4. Modify the program to create a macro named **StormChart**. Replace the three %LET statements with positional parameters for **Basin**, **Season**, and **MaxWind**. Add an OPTIONS statement to change the MCOMPILENOTE= option to ALL. Run the program and verify that the macro compiled without error.

```sas
options mcompilenote=all;
%macro stormchart(Basin, Season, MaxWind);
title1 "Storm Frequency by Type";
title2 "&Basin Basin, &Season Season, Max Wind > &MaxWind MPH";
... title;
%mend stormchart;
```
5. Call the %StormChart macro with parameter values \textit{EP} for \textbf{Basin}, \textit{2015} for \textbf{Season}, and \textit{125} for \textbf{MaxWind}. Turn on the MPRINT option to view the submitted macro code in the log and turn off the MPRINT option after the macro executes. Highlight the statements and run the selected code. Review the results to verify that the results match the previous report. Confirm in the log that the executed code is printed.

```sas
options mprint;
%stormchart(EP,2015,125)
options nomprint;
```

End of Demonstration
4.1 Defining and Calling a Macro

Macro Positional Parameters

```sas
%macro printable(dsn,obs);
proc print data=&dsn(obs=&obs);
run;
%mend printable;

%mprint(printable(sashelp.cars))
```

If a positional parameter is not assigned a value in the macro call, it is assigned a null value, which can cause syntax errors.

```sas
75 proc print data=&dsn(obs=&obs); run;
...
%MPRINT(PRINTTABLE): proc print data=sashelp.cars(obs=); run;
%MPRINT(PRINTTABLE): run;
ERROR 23-7: Invalid value for the OBS option.
```
4.1 Defining and Calling a Macro

Macro Keyword Parameters

%MACRO macro-name(name=value, ..., name=value);

%macro printable(dsn=sashelp.cars,obs=5);

Keyword parameters in a macro definition are listed as name=value pairs.

Keyword parameters enable you to assign default values.
4.1 Defining and Calling a Macro

Macro Keyword Parameters

\texttt{%macro-name(name=value, \ldots, name=value)}

Keyword parameters in a macro call
• include \texttt{name=value} pairs
• can be listed in any order
• require parentheses after the macro name even if no parameter values are provided.
If keyword parameters are omitted, the default value from the macro definition is assigned.
Open `m104p05.sas` from the `activities` folder and perform the following tasks:

1. Modify the `%MACRO` statement to use keyword parameters. Assign default values of `NA` to `Basin`, `2016` to `Season`, and `20` to `MaxWind`. Run the program and verify that the macro compiles successfully.

   ```sas
   %macro stormchart(basin=NA, season=2016, maxwind=20);
   %stormchart()
   ```

2. Call the `%StormChart` macro with no parameter values provided. Confirm in the log that the default parameter values are used.

   ```sas
   %stormchart()
   ```

3. Call the `%StormChart` macro with `Season` as `2015` and `Basin` as `EP`. View the log. Is the table subset by `MaxWind`, even though it is not included as a parameter in the macro call?
4.1 Defining and Calling a Macro

4.03 Activity – Correct Answer

Is the table subset by MaxWind, even though it is not included as a parameter in the macro call? **Yes**

```latex
\%stormchart(season=2015, basin=EP)
```

The default value for the **MaxWind** parameter is used in the WHERE statement.

**NOTE:** There were 26 observations read from the data set MC1.STORM_FINAL. WHERE (Basin='EP') and (Season=2015) and (MaxWindMPH>20);
4.1 Defining and Calling a Macro

Mixed Parameter Lists

%macro stormchart(Basin, Season=2016, MaxWind=111);

%stormchart(WP)
%stormchart(NA, Season=2015)
%stormchart(SI, MaxWind=100, Season=2014)

It is possible to mix positional and keyword parameters in a list, but positional parameters must always come first.
4.1 Defining and Calling a Macro

Mixed Parameter Lists

```
%macro stormchart(Basin, Season=2016, MaxWind=111);

%stormchart(WP)

%stormchart(NA, Season=2015)

%stormchart(SI, MaxWind=100, Season=2014)
```

_Basin_ does not have a default value as a positional parameter, so a value must be provided in each macro call.
## Lesson 4: Working with Macro Programs

<table>
<thead>
<tr>
<th>4.1 Defining and Calling a Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2 Conditional Processing</td>
</tr>
<tr>
<td>4.3 Iterative Processing</td>
</tr>
</tbody>
</table>
4.2 Conditional Processing

data Europe USA Asia;
  set sashelp.cars;
  if Origin="Europe" then output Europe;
  else if Origin="USA" then output USA;
  else output Asia;
run;

IF-THEN statements in the DATA step enable us to execute code conditionally for each row processed in the data.
data sports;
  set sashelp.cars;
  where Type="Sports";
  AvgMPG=mean(MPG_City, MPG_Highway);
run;

title "Sports Cars";
proc print data=sports noobs;
  var Make Model AvgMPG MSRP EngineSize;
run;
title;
proc sgplot data=sports;
  scatter x=MSRP y=EngineSize;
run;

How can I compile and execute parts of my program conditionally?

IF the DATA step runs successfully THEN run the rest of the program
4.2 Conditional Processing

Conditional Processing of Text

Macro conditional statements can be used either inside or outside a macro definition.
4.2 Conditional Processing

Conditional Processing of Text

%IF expression %THEN action-1;
%ELSE action-2;

%IF expression %THEN %DO;
  action-1
%END;
%ELSE %IF expression %THEN %DO;
  action-2
%END;
%ELSE %DO;
  action-3
%END;

- %IF-%THEN can be used to execute one statement.
- Use a %DO group to execute more than one statement conditionally.
- %ELSE %IF nested conditions are allowed.
- %ELSE or %ELSE %DO is optional.
4.2 Conditional Processing

Conditional Processing of Text

%IF expression %THEN %DO;
    action-1
%END;
%ELSE %DO;
    action-2
%END;

- must use a %DO group
- cannot use %ELSE %IF nested conditions
- %ELSE %DO is optional.

outside a macro definition
data sports;
   set sashelp.cars;
   where Type="Sports";
   AvgMPG=mean(MPG_City, MPG_Highway);
run;

title "Sports Cars";
proc print data=sports noobs;
   var Make Model AvgMPG MSRP EngineSize;
run;

proc sgplot data=sports;
   scatter x=MSRP y=EngineSize;
run;

If the DATA step runs without error, compile and execute the rest of the program.

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>AvgMPG</th>
<th>MSRP</th>
<th>EngineSize</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acura</td>
<td>NSX coupe 2Dr Manual S</td>
<td>20.5</td>
<td>$50,705</td>
<td>3.2</td>
</tr>
<tr>
<td>Audi</td>
<td>RS 6 4Dr</td>
<td>10.5</td>
<td>$84,600</td>
<td>4.0</td>
</tr>
<tr>
<td>Audi</td>
<td>TT 1.8 convertible 2Dr (coupe)</td>
<td>24.0</td>
<td>$35,940</td>
<td>1.8</td>
</tr>
<tr>
<td>Audi</td>
<td>TT 1.8 Quattro 2Dr (convertible)</td>
<td>24.0</td>
<td>$37,380</td>
<td>1.8</td>
</tr>
<tr>
<td>Audi</td>
<td>TT 3.2 coupe 2Dr (convertible)</td>
<td>25.0</td>
<td>$40,550</td>
<td>3.2</td>
</tr>
<tr>
<td>BMW</td>
<td>650 convertible 2Dr</td>
<td>20.0</td>
<td>$48,195</td>
<td>3.0</td>
</tr>
<tr>
<td>BMW</td>
<td>30i convertible 2Dr</td>
<td>19.5</td>
<td>$35,595</td>
<td>2.2</td>
</tr>
<tr>
<td>BMW</td>
<td>325i convertible 2Dr</td>
<td>22.0</td>
<td>$32,000</td>
<td>2.5</td>
</tr>
<tr>
<td>BMW</td>
<td>328i convertible 2Dr</td>
<td>25.0</td>
<td>$34,545</td>
<td>3.0</td>
</tr>
<tr>
<td>BMW</td>
<td>335i convertible 2Dr</td>
<td>21.0</td>
<td>$36,750</td>
<td>4.0</td>
</tr>
</tbody>
</table>
If the DATA step generates an error, write a custom error message to the log and do not execute the rest of the program.

**ERROR**: Libref SASHLP is not assigned.
**ERROR**: The rest of the program will not execute.
data sports;
  set sashlp.cars;
  where Type="Sports";
  AvgMPG=mean(MPG_City, MPG_Highway);
run;

title "Sports Cars";
proc print data=sports noobs;
  var Make Model AvgMPG MSRP EngineSize;
run;

SYSERR=0 if the step ran without errors. SYSERR≠0 if the step generated an error.
**4.2 Conditional Processing**

Conditional Processing of Text

```sas
data sports;
  set sashlp.cars;
  where Type="Sports";
  AvgMPG=mean(MPG_City, MPG_Highway);
run;

%if &syserr ne 0 %then %do;
  %put ERROR: The rest of the program will not execute;
%end;
%else %do;
  title "Sports Cars";
  proc print data=sports noobs;
  ...
  run;
%end;
```
data sports;
    set sashelp.cars;
    where Type="Sports";
    AvgMPG=mean(MPG_City, MPG_Highway);
run;

%if &syserr ne 0 %then %do;
    %put ERROR: The rest of the program will not execute;
%end;
%else %do;
    title "Sports Cars";
    proc print data=sports noobs;
    ...
run;
%end;
Open m104p07.sas from the activities folder and perform the following tasks:

1. Notice that there is a missing semicolon after the DATA statement, which causes a syntax error. If there is a syntax error in the first DATA step, then the %PUT statement should execute and write a custom error message to the log, and the remaining PROC steps should not execute.

2. Run the program and look at the log. Notice that although there was an error in the DATA step, SAS still attempted to execute PROC PRINT and PROC SGPLOT.

3. Identify the syntax error in the macro %IF statement. Fix the error and run the program again to confirm that the %PUT statement is executed.
Identify the syntax error in the macro %IF statement. Fix the error and run the program again to confirm that the %PUT statement is executed.

```
ERROR: Expected %THEN statement not found.
78 79 %if &syserr ne 0 then do;
80 81 %put ERROR: The rest of the program will not execute;
81 %end;
```

Be sure to always include the percent sign in front of macro keywords %IF, %THEN, %ELSE, and %DO.
We want to calculate fuel efficiency differently depending on the value of **Origin**.
Scenario: Nested Conditional Processing

```sas
if &loc is null
   %put ERROR: Provide a value for Origin.;
   %put ERROR- Valid values: Asia, Europe, USA;
else if &loc is USA
   data fuel_&loc;
      set sashelp.cars;
      where Origin="&loc";
      AvgMPG=mean(MPG_City, MPG_Highway);
      keep Make Model Type AvgMPG;
   run;
else if &loc is Asia or Europe
   data fuel_&loc;
      set sashelp.cars;
      where Origin="&loc";
      AvgKmL=mean(MPG_City, MPG_Highway)*.425;
      keep Make Model Type AvgKmL;
   run;
```

AvgMPG: Average miles per gallon
AvgKmL: Average kilometers per liter
Scenario: Nested Conditional Processing

%macro avgfuel(loc);

if &loc is null
else if &loc is USA
else if &loc is Asia or Europe

%mend avgfuel;

Nested conditional steps must be included inside a macro definition.
It is a good practice to account for possible null values.

&macro-var= &macro-var EQ

&macro-var ^= &macro-var NE

%macro avgfuel(loc);
%if &loc= %then %do;
  %put ERROR: Provide a value for Origin.;
  %put ERROR- Valid values: Asia, Europe, USA;
%end;
...

#PharmaSUG2021
%macro avgfuel(loc);
%if &loc= %then %do;
  %put ERROR: Provide a value for Origin.;
  %put ERROR- Valid values: Asia, Europe, USA;
  %return;
%end;
...
proc print data=fuel_&loc;
run;
%RETURN;

terminate macro execution
%macro avgfuel(loc);
...
%else %if &loc=USA %then %do;
  data fuel_&loc;
  set sashelp.cars;
  where Origin="&loc";
  AvgMPG=mean(MPG_City, MPG_Highway);
  keep Make Model_Type AvgMPG;
  run;
%end;
...

Remember that text is not quoted in macro comparisons.

If &loc is USA, this step is tokenized, compiled, and executed.
...%else %do;
    data fuel_&loc;
    set sashelp.cars;
    where Origin="&loc";
    AvgKmL=mean(MPG_City, MPG_Highway)*.425;
    keep Make Model Type AvgKmL;
    run;
%end;
%mend avgfuel;

If \&loc\ is not null or USA, this step is tokenized, compiled, and executed.
Open `m104p08.sas` from the `activities` folder and perform the following tasks:

1. Review the program and notice that different DATA steps run, depending on the value of the `loc` parameter. The `%avgfuel` macro call at the end of the program assigns `Europe` to the `loc` parameter.

2. Run the program and view the log. Notice the MPRINT messages displaying the final program that was compiled and executed.

3. Call the `%avgfuel` macro with `usa` as the parameter value. View the log. Which DATA step was compiled and executed?
Which DATA step was compiled and executed?

```sas
%avgfuel(usa)
74 %avgfuel(usa)
MPRINT(AVGFUEL): data fuel_usa;
MPRINT(AVGFUEL): set sashelp.cars;
MPRINT(AVGFUEL): where Origin="usa";
MPRINT(AVGFUEL): AvgKmL=mean(MPG_City, MPG_Highway)*.425;
MPRINT(AVGFUEL): keep Make Model Type AvgKmL;
MPRINT(AVGFUEL): run;
```

The second DATA step executes.
Lesson 4: Working with Macro Programs

4.1 Defining and Calling a Macro

4.2 Conditional Processing

4.3 Iterative Processing
title "3-Cylinder Cars";
proc print data=sashelp.cars noobs;
   where Cylinders=3;
   var Cylinders Make Model Type Origin MSRP MPG_City MPG_Highway;
run;

<table>
<thead>
<tr>
<th>Cylinders</th>
<th>Make</th>
<th>Model</th>
<th>Type</th>
<th>Origin</th>
<th>MSRP</th>
<th>MPG_City</th>
<th>MPG_Highway</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Honda</td>
<td>Insight 2dr (gas/electric)</td>
<td>Hybrid</td>
<td>Asia</td>
<td>$19,110</td>
<td>60</td>
<td>66</td>
</tr>
</tbody>
</table>
4.3 Iterative Processing
Developing Macro Applications

- Start with a validated SAS program
- Generalize with macro variables
- Create a macro definition with parameters
- Use macro-level programming for complex processing
- Add Data Driven Features
- Validate parameters and document

%let num=3;
%let num=3;

title "&num-Cylinder Cars";
proc print data=sashelp.cars noobs;
  where Cylinders=&num;
  var Make Model Type Origin MSRP MPG_City MPG_Highway;
run;

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Type</th>
<th>Origin</th>
<th>MSRP</th>
<th>MPG_City</th>
<th>MPG_Highway</th>
</tr>
</thead>
<tbody>
<tr>
<td>Honda</td>
<td>Insight 2dr (gas/electric)</td>
<td>Hybrid</td>
<td>Asia</td>
<td>$19,110</td>
<td>60</td>
<td>66</td>
</tr>
</tbody>
</table>
4.3 Iterative Processing
Developing Macro Applications

- Start with a validated SAS program
- Generalize with macro variables
- Create a macro definition with parameters
- Use macro-level programming for complex processing
- Add Data Driven Features
- Validate parameters and document

```sas
%macro cylreport(num);
  title "&num-Cylinder Cars";
  proc print data=sashelp.cars noobs;
    where Cylinders=&num;
    var Make Model Type Origin MSRP MPG_City MPG_Highway;
  run;
%mend cylreport;
%cylreport(3)
```

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Type</th>
<th>Origin</th>
<th>MSRP</th>
<th>MPG_City</th>
<th>MPG_Highway</th>
</tr>
</thead>
<tbody>
<tr>
<td>Honda</td>
<td>Insight 2dr (gas/electric)</td>
<td>Hybrid</td>
<td>Asia</td>
<td>$19,110</td>
<td>60</td>
<td>66</td>
</tr>
</tbody>
</table>

#PharmaSUG2021
%macro cylreport(num);
    title "&num-Cylinder Cars";
    proc print data=sashelp.cars noobs;
        where Cylinders=&num;
        var Make Model Type Origin MSRP MPG_City MPG_Highway;
    run;
%mend cylreport;

%cylreport(3)
4.3 Iterative Processing

%DO Iterative Statement

%DO index=start %TO stop <%BY increment>;
text and macro language statements
%END;

- *index* is a macro variable.
- *start*, *stop*, and *increment* must be integer values.
- The default for *increment* is 1.
- *index* is incremented at %END.
- At the top of the loop, if *index* > *stop*, the loop terminates.

%DO loops are valid only within a macro definition.
%macro allcylinders(start,stop);
  %do num=&start %to &stop;
    title "&num-Cylinder Cars";
    proc print data=sashelp.cars noobs;
      where Cylinders=&num;
      var Make Model Type Origin
      MSRP MPG_City MPG_Highway;
    run;
  %end;
%mend allcylinders;

%allcylinders(3,6)

Use a %DO loop inside the macro to execute the program for each value of Cylinder, 3 through 6.
4.3 Iterative Processing

**%DO Iterative Statement**

85 %allcylinders(3,6)

MLOGIC(ALLCYLINDERS): Beginning execution.
MLOGIC(ALLCYLINDERS): Parameter START has value 3
MLOGIC(ALLCYLINDERS): Parameter STOP has value 6
MLOGIC(ALLCYLINDERS): %DO loop beginning; index variable NUM; start value is 3; stop value is 6; by value is 1.

MPRINT(ALLCYLINDERS): title "3-Cylinder Cars";
MPRINT(ALLCYLINDERS): proc print data=sashelp.cars noobs;
MPRINT(ALLCYLINDERS): where Cylinders=3;
MPRINT(ALLCYLINDERS): var Make Model Type Origin MSRP MPG_City MPG_Highway;
MPRINT(ALLCYLINDERS): run;
NOTE: There were 1 observations read from the data set SASHELP.CARS.
WHERE Cylinders=3;
...
MLOGIC(ALLCYLINDERS): %DO loop index variable NUM is now 4; loop will iterate again.
MPRINT(ALLCYLINDERS): title "4-Cylinder Cars";
...

4.3 Iterative Processing

%DO Iterative Statement

create a driver macro to call the %CyLReport macro once for each value, 3 through 6.
Iterative %DO loops

This demonstration illustrates using macro %DO iterative processing.
Iterative %DO Loops

Scenario

In this demonstration, we use macro %DO iterative processing to produce the storm report for a range of years.

Files

m104p10.sas

Syntax

```
%DO index = start %TO stop;
  text to be generated
%END;
```

Notes

- Iterative processing can be used to generate repetitive program statements.
- A macro can call another macro for execution.

Demo

1. Open m104p10.sas from the demos folder. This program creates the %StormChart macro definition, which accepts two parameters: Basin and Season. The macro generates a bar chart representing the number of storms by storm type for the selected Basin and Season.

2. Run the macro definition and the macro call with Basin as NA and Season as 2015. Verify that the bar chart is created and includes 12 storms.

3. Create a driver macro named StormChartRange that calls the %StormChart macro for a range of years.
   a. Start with a %MACRO statement that includes the parameters Basin, Start, and Stop.
   b. Add a %DO loop with an index variable named Season that ranges from the Start macro variable value to the Stop macro variable value. Inside the %DO loop, call the %StormChart macro with the parameter values provided by Basin and Season.
   c. Add a %MEND statement to complete the macro definition.
   d. Compile the %StormChartRange macro.

```
%macro stormchartrange (Basin, Start, Stop);
%local Season;
  %do Season=&Start %to &Stop;
    %stormchart (&Basin, &Season)
  %end;
%mend stormchartrange;
```
4. Test the `%StormChartRange` macro, generating reports for a range of seasons for different basins.
   
   a. Submit the following macro call and verify the results:
      
      ```
      %StormChartRange( EP, 2011, 2013)
      ```
      
   b. Submit the following macro call and verify the results:
      
      ```
      %StormChartRange( NA, 2010, 2016)
      ```

   End of Demonstration
Section 5: Store A SQL Query In A Macro

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Using SQL to Create Macro Variables</td>
</tr>
<tr>
<td>3.3</td>
<td>Using the DATA Step to Create Macro Variables</td>
</tr>
<tr>
<td>3.4</td>
<td>Indirect References to Macro Variables</td>
</tr>
</tbody>
</table>
%LET is an easy way to create a macro variable and assign a value.

%let type=Sedan;
PROC SQL can create and assign macro variables based on your data.
PROC SQL;
SELECT <DISTINCT> item-1 <, item-2, ...>
FROM clause
WHERE clause
ORDER BY clause;
QUIT;

PROC SQL;
proc sql;
select Model, MPG_Highway
from sashelp.cars
where MPG_Highway>50
order by MPG_Highway;
quit;

<table>
<thead>
<tr>
<th>Model</th>
<th>MPG (Highway)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prius 4dr (gas/electric)</td>
<td>51</td>
</tr>
<tr>
<td>Civic Hybrid 4dr manual (gas/electric)</td>
<td>51</td>
</tr>
<tr>
<td>Insight 2dr (gas/electric)</td>
<td>66</td>
</tr>
</tbody>
</table>
5.1 Syntax 1- Store a SQL Query in a Macro
Creating Macro Variables with PROC SQL

PROC SQL;
SELECT <DISTINCT> item-1 <, item-2, ...>
   <INTO : macvar-1 < ..., : macvar-n>
FROM clause
   <WHERE clause>
   <ORDER BY clause>;
QUIT;

The INTO clause assigns values produced by the query to macro variables.

Be sure to precede each macro variable name with a colon.
5.1 Syntax 1- Store a SQL Query in a Macro

Step 1 – Create A Macro

```sql
proc sql;
select (*)
  into:total
from sashelp.cars;
quit;

%put &total &sqlobs;
```
Syntax 1 - Macro Variables with PROC SQL

```sas
proc sql;
select count(*)
into: total
from sashelp.cars;
quit;
```

Count the number of rows returned by the query and store in a macro variable named **Total**.

Leading spaces are included in the value.

SQLOBS is created automatically and stores the number of rows returned by the query.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL</td>
<td>428</td>
</tr>
<tr>
<td>SQLOBS</td>
<td>1</td>
</tr>
</tbody>
</table>
Syntax 1 - Macro Variables with PROC SQL

```sas
proc sql;
select count(*)
into :total trimmed
from sashelp.cars;
quit;
```

Use TRIMMED to remove spaces before values are stored in the macro variable.

Global Symbol Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL</td>
<td>428</td>
</tr>
<tr>
<td>SQLOBS</td>
<td>1</td>
</tr>
</tbody>
</table>
Syntax 2: Macro Variables with PROC SQL

```
proc sql noprint;
select distinct Origin into :origin1-:origin3
from sashelp.cars;
quit;
```

- Suppresses the report
- Creates a series of macro variables for the three distinct values of `Origin`

**Global Symbol Table**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORIGIN1</td>
<td>Asia</td>
</tr>
<tr>
<td>ORIGIN2</td>
<td>Europe</td>
</tr>
<tr>
<td>ORIGIN3</td>
<td>USA</td>
</tr>
<tr>
<td>SQLOBS</td>
<td>3</td>
</tr>
</tbody>
</table>
Syntax 2: Macro Variables with PROC SQL

```
proc sql noprint;
select distinct Type
into :type1-
from sashelp.cars;
quit;
```

If you don’t know how many macro variables to create, you can omit the upper bound.

Global Symbol Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE1</td>
<td>Hybrid</td>
</tr>
<tr>
<td>TYPE2</td>
<td>SUV</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>TYPE6</td>
<td>Wagon</td>
</tr>
<tr>
<td>SQLOBS</td>
<td>6</td>
</tr>
</tbody>
</table>
Syntax 3: Macro Variables with PROC SQL

```sas
proc sql noprint;
select distinct Origin
  into :originlist separated by "", "
from sashelp.cars;
quit;
```

Global Symbol Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORIGINLIST</td>
<td>Asia, Europe, USA</td>
</tr>
<tr>
<td>SQLOBS</td>
<td>3</td>
</tr>
</tbody>
</table>

Use SEPARATED BY to assign multiple values to a single macro variable.
Questions?
6. Handy Links

- [SAS® Macro Programming for Beginners](#)—Susan J. Slaughter, Lora D. Delwiche
- [SAS® 9.4 Macro Language: Reference, Fifth Edition](#)
- [Getting things in order with PROC SQL, macro and a SAS function](#)—Charu Shankar
Name: Charu Shankar
Affiliation: SAS Institute Inc.
Contact Number:
E-mail: Charutraining@gmail.com
Website: https://blogs.sas.com/content/author/charushankar/
Twitter: Charuyogacan
LinkedIn: www.linkedin.com/in/charushankar