Topic – Using SAS Macros for efficient Data Manipulation

- By Praveen Tiwari

Introduction

If you've been writing the same lines of code repeatedly in SAS, you can stop now. It shouldn't be as laborious as you've been. Availability of SAS Macros can make your work faster and save your day. I've worked on SAS for 3 years. I remember writing same lines of code every morning, until I learnt Macros. Since then, I've persisted to learn every bit of technique to make coding faster in SAS.

Macros provide an incredible way to automate a process. Over the years, I've learnt these ways which I've shared in this article. Macros are simple to learn but can be confusing at times. Hence, you must pay attention to the command and its output. Lots of people end of mixing the two.

Here is a complete tutorial on using SAS Macros. I hope this will help you to become better and faster in SAS.

The topics covered are as shown below.

Table of Contents

Initiate your SAS software and let's get started. Performing these commands as you read through would help you memorize these commands better.

1. Introduction to SAS Macros

I'll take a simple example so that you can understand this concept better:

Example:

Let's look at the following SAS program:

```
proc print data=Sales.Policysnapshot;


where policydate='09Sep14'd;
```

```
title “Below are the policy detail for 09-Sep-14”;


Run;
```

Above SAS code is written to extract policy level details for 09-Sep-14 and let us say that the user needs to run this code on a daily basis after changing the date (at both the places) to current date.

Thankfully, SAS – like any other programming language, provides the facility of automating codes with use of SAS macro programming. Now, let us look at the same code once automated by SAS macros.

```
Proc Print Data=Sales.Policysnapshot;


Where policydate=”&sysdate9”d;


Title “Below are the policy detail for %sysdate9”;


Run;
```

In this case, the user does not need to provide value for current date. The code will automatically pick current date from system.

What are the benefits of using SAS Macros ?

SAS macros enable us to:-

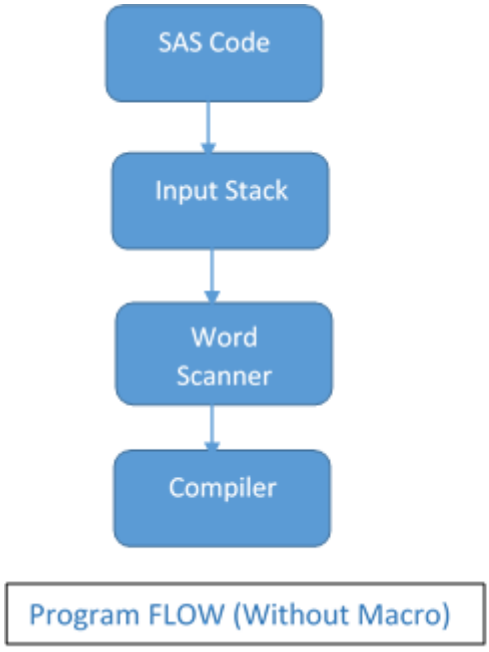
- Make our job easier by re-using similar code multiple times after defining it ones.
- Make changes in variable at a single place and reflect them at multiple locations.
- Make your programs data driven, i.e. letting SAS decide what to do based on actual data values

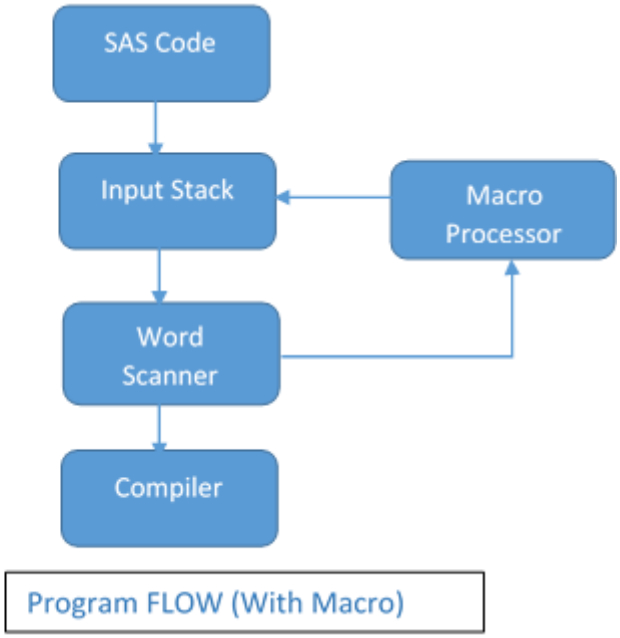
Above all, it helps to reduce the efforts required to read and write SAS Code. Macro programming is typically covered as advanced topic in SAS, but the basic concepts of SAS macros are easy to understand.

I will now introduce the concept of SAS programming and I assume that you are aware about basics of SAS programming. We will look at how macro processor works, how to use SAS macros & macro variables and How to create flexible and reusable code to save time and efforts?

How does Macro Processor Work?

A SAS program is a combination of Data steps, global statements, SAS Component Language (SCL), SQL statements and SAS Macro statements. Whenever we submit a program, it gets copied in memory (called input stack) followed by word scanner and there after it goes to compiler and gets executed. When we use Macros in our SAS program, an additional step is added to resolve macros. This step is called as MACRO Processor.





As we can see above, we have an additional step “Macro Processor” when we work with macro i.e. SAS code with macro does not compile or execute faster. It only helps to reuse the SAS code and variables values.

What are the components of Macro code?

Macro code consists of two basic building blocks: Macros and Macro variables. In a SAS program they are referred differently as:-

- &Name refers to Macro Variable
- %Name refers to Macro

What are Macro Variables?

A macro variable is just like a standard variable, except that its value is not a data set and has only a single character value. The value of a macro variable could be a variable name, a numeral, or any text you want substituted in your program.

Scope of Macro variable can be local or global variables on how we have defined it. If it is defined inside a macro program, then scope is local (only available for that macro code). However, if we have defined it outside (in the main body), then we can use it anywhere in SAS program.

%LET statement is used to create and assign value to macro variables.

% LET <Macro Variable Name>=Value;

Macro variable name follows the SAS naming convention and if variable already exists then value is overwritten.

Value of macro variable in %Let statement can be any string and it has following characteristics:-

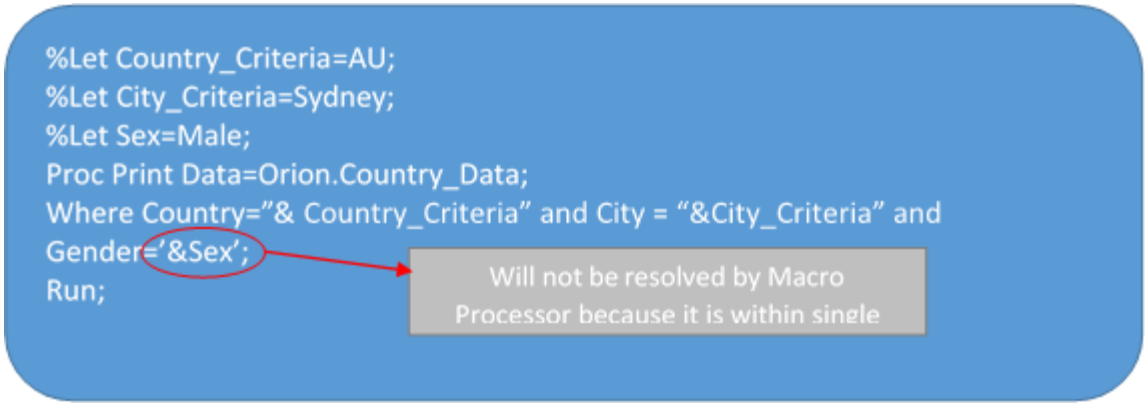
- It can be of any length between 0 to 65,534 characters
- Numeric values are also stored as character string
- Mathematical expressions are not evaluated
- Quotation mark can also be stored as a part of value
- Leading and trailing blanks are removed before assignment

Macro variables are referenced by using ampersand (&) followed by macro variable name.

&<Macro variable Name>

Example:-

We can also declare multiple macro variables and use them at different places in SAS Code. Macro variable are not resolved when they are accessed within single quotes. Hence, we should use double quotes to reference them.



How to reference a Macro variable with text ?

We can reference macro variable with text in multiple ways:-

- Text**&macrovaribale**

- **&macrovariable**Text
- Text**&macrovraible**Text

Let's look at these with a simple example:-

SAS Programs

```
%Let Year=2014;
%Let Month=SEP;
Proc Print data = Test.&Year&MonthEND
; Run;
```

After Macro processor evaluation

```
%Let Year=2014;
%Let Month=SEP;
Proc Print data = Test.2000SEPEnd;
Run;
```

A period (.) is used as delimiter that defines the end of a macro variable.

Let's look at an example:-

```
%Let Lib_name=Orion;

%Let dataset_name=Country;

Proc Print data = &Lib_name.&dataset_name;

Run;
```

```
%Let Lib_name=Test;

%Let dataset_name=Sale;

Proc Print data = ORIONCOUNTRY;

Run;
```

Here, you can see that the program did not execute because we required (ORION.COUNTRY) and not (ORIONCOUNTRY). This happened because here period (.) is used as separator between two macro variables. Now, to use period as a separator between library name and dataset, we need to provide period (.) twice.

```
%Let Lib_name=Orion;

%Let dataset_name=Country;

Proc Print data = &Lib_name..&dataset_name;

Run;
```

SAS Macros

SAS Macros are useful when we want to execute same set of SAS statements again and again. This is an ideal case for use of macro rather than typing / copy pasting same statements.

Macro is a group of SAS statements that is identified by a name and to use it in program anywhere, we can reference it with that name.

Syntax:-

%Macro <Macro Name>;

Macro Statements;

%MEND;

Macro Statements can contain combination of the following:-

- Text
- SAS statements or steps
- Macro variable references
- Macro statements, expressions or calls

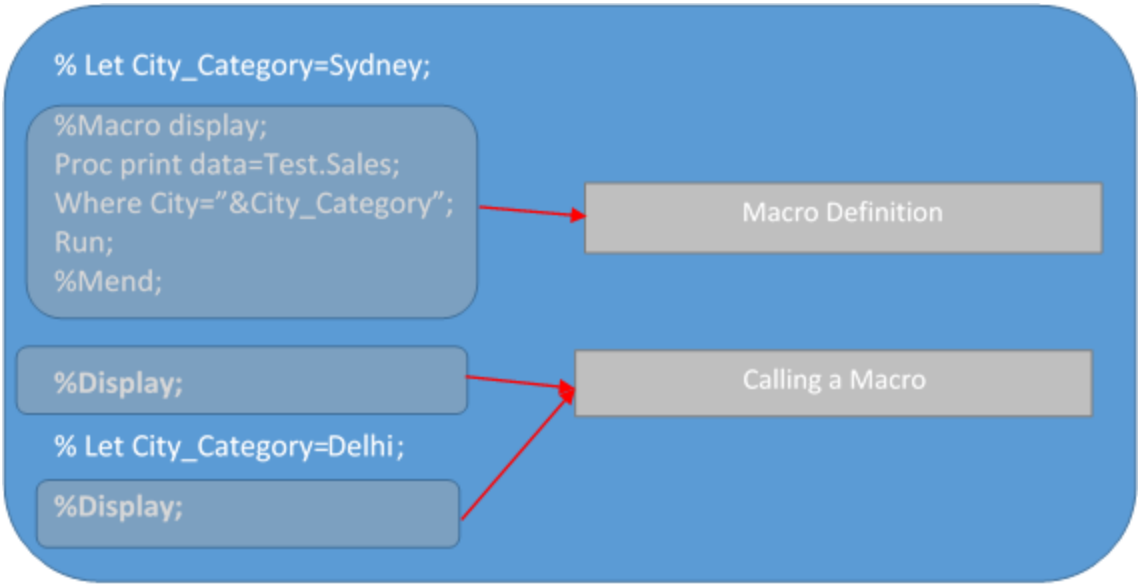
After definition of macro, we need to invoke/ call it when want to use it.

Syntax for calling a MACRO: –

%<Macro Name> [;]

Here we do not need to end this statement with a semicolon but it is a good programming practice to have it.

Example:-



You can see that we have used same set of statements twice using macro.

Adding parameter to SAS Macros

We can define a Macro with parameters, which can be referenced within the macro. We can pass the parameters in two ways:-

- Positional Parameters
- Keyword Parameters

Positional Parameters: In this method, we supply parameter name at time of defining the macro and values are passed at time of Macro call.

Syntax:-

Definition

```
%MACRO <macro name>(Parameter1, Parameter2,….Parameter-n);


Macro text;


%MEND;
```

Calling

```
%MacroName (Value1, Value2,…..Value-n);
```

At calling stage, values to parameters are passed in similar order as they are defined in the macro definition.

Example:-

```
%MACRO AV(Country_Criteria, City_Criteria);


Proc Print data=Test.sales;


Where Country=”&Country_Criteria” AND CITY=”&City_Criteria”;


Run;


%AV(AU, Sydney);
```

Keyword Parameters: In this method, we provide parameter name with equals to sign and also can assign default value to the parameters. While calling the macro, we have to mention the parameter names followed by equals sign and value.

In this method, parameters with equals sign can be

- Specified in any order
- Omitted from the call (In this case, the macro will use default value given at definition)

Syntax:-

Definition

```
%MACRO <macro name> (Parameter1=Value, Parameter2=Value…….Parameter-n=Value);


Macro Text;


%MEND;
```

Calling

```
%<macro name> (Parameter1=Value, Parameter2=Value …..Parameter-n=Value);
```

Example:-

```
%MACRO AV (Country_Criteria=IND, Gender=Male);
Proc Print data=Test.sales;
Where Country="&Country_Criteria" AND Sex="&Gender";
Run;


%AV(Country_Criteria=AU, Sex=Female); -- Will display records for Country Australia and Sex Female.
%AV(Country_Criteria=Delhi, Sex=Male); -- Will display records for Country India and Sex Male.
%AV(City_Criteria=UK); -- Will display records for Country UK and Sex Male (default value of Sex).
```

Let's discuss some examples of SAS Macro Variable and SAS Macros, when you want to use a particular value multiple times in a program and its value changes over time.

For example:-

- You want to extract transaction details, Manpower details, Budget details and others for current month only and this is monthly task (Repetitive). You can provide value for current month (between 01-Sep-14 to 30-Sep-14) and with help of Macro Variables, we can create the programme in such a manner that we need to define them only once.

- At initial stage of my career, I used to write SAS codes for data conversion like text to number, number to text, adjusting width of the data type and converting report in specific layout format and others. But with the help of SAS macros, I can write SAS macro for each specific task and call it whenever they are required.

SAS Macros are typically considered as part of advance SAS Programming and are used widely in reporting, data manipulation and automation of SAS programs. They do not help to reduce the time of execution, but instead, they reduce repetition of similar steps in your program and enhance the readability of Programs.

Let's now proceed further can learn the process of creating conditional and repetitive steps using SAS Macro.

2. SAS Macros – Conditional and Iterative statement

SAS macros provide us with the flexibility to use a piece of code multiple times by simply calling the macro. This flexibility can be exploited to reach next level of sophistication with use of conditional statements and loops using macro statements such as %IF, %DO. These macro statements should always be called inside the macro.

What is Conditional Processing?

As the name implies, conditional processing is used when we want to execute a piece of code based on the output of single or multiple conditions.

Syntax:-

```
%IF <Conditional Macro Expression> %THEN <TRUE Text>;
%ELSE <FALSE Text>;

                    Or

%IF <Conditional Macro Expression> %THEN %DO;
Text1; Text2 ... Text-n;
%END;
%ELSE %DO;
Text1; Text2 ... Text-n;
%END;
```

Here, ELSE part of the syntax is optional.

Both syntax work in similar manner – the only difference is, first one executes only one statement after THEN or ELSE, whereas the second syntax can execute multiple statements.

What are Conditional Macro Expression?

I am referring to the expression in the condition as Conditional macro expression. In some cases, it is similar to SAS expression:

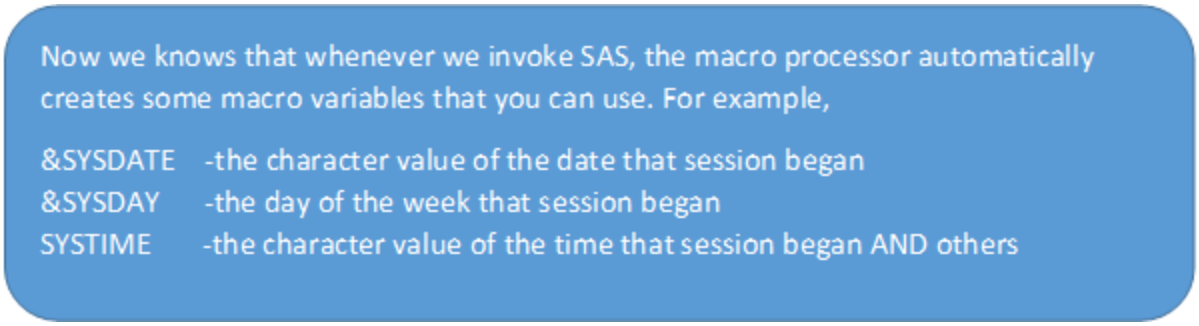
1. Arithmetic operators
2. Logical Operators (do not use %sign before AND or OR)
3. Comparison operators
4. It is also case sensitive

Dissimilarities compared to SAS Expression:

1. Character operands are not quoted (Single or double)
2. Ranges 20000 <= &Salary <=5000 do not work, we should write it as &Salary >= 20000 AND &Salary <=50000
3. IN operator not requires parentheses.

Example

Let's say, we have transactional data for a company which deals in food business. Here manager requires details of all the sales on daily basis except on MONDAY. On Monday, he want a additional summary report for item wise total sales (People call it as MONDAY Report!)



Here we will first define a separate macro for Daily and Monday reports (you can combine these together also).

```
%MACRO Daily;
PROC PRINT DATA=Sale.Transaction;
TITLE "Daily Report: Sales Detail";
RUN;
%MEND;

%MACRO Monday;
PROC MEANS DATA=Sale. Transaction MEAN MIN MAX;
CLASS Item;
VAR Sales;
TITLE "Monday Report: Summary of ITEM wise Total Sales";
RUN;
%MEND;

%Macro Reporting;
%Daily;
%IF &SYSDAY=Monday %THEN %Monday;
%Mend;

%Reporting;
```

Now on daily basis it will call Macro "Daily" and check if today is MONDAY then it will run MACRO "Monday". Similarly if we have multiple statements to execute after %IF-%Then then we should go for %IF-%THEN-%DO-%END.

Loops

Loops are used to create a dynamic program, which executes for a number of iterations, based on some conditions (called conditional iteration). These statements are also valid only inside a macro.

Syntax:

```
%DO <index-variable>=<start> %to <stop> [%BY increment value];
Text1;
Text2;
.
.
Text-n;
%END;
```

%DO %WHILE <Expression>; Text1; Text2; . . Text-n; %END;	%DO %UNTIL <Expression>; Text1; Text2; . . Text-n; %END;

Key things about syntax:

1. Index variable should be a macro variable.
2. Start, Stop and Increment Value can be any macro expression that resolves to integers.

3. %BY statement is optional (default increment value is 1).
4. %While and %Until work similar, basic difference between these two is %WHILE first evaluates the criteria then executes loop whereas %UNTIL first executes loop i.e. %UNTIL evaluate at least once.

Example:

Let's say we have a series of SAS data sets YR1990 – YR2013 that contain business detail and now we want to calculate the average sales for each of these years.

If we will not use the SAS macro then we need to write PROC MEANS steps for each of the year, which is not a good idea.

Now let's look at how we can use SAS Macros to do this job easily:

```
%MACRO CalcAverage;

%DO i = 1990 %TO 2013;

PROC MEANS DATA=YR&i;
VAR SCORE;
TITLE "Average math score of Class &I";
RUN;

%END;

%MEND CalcAverage;

% CalcAverage;
```

We can achieve this with %UNTIL and %WHILE.

Conditional statements and loops empower SAS Macros to perform conditional and iterative tasks. They appear similar to SAS conditional statements and loops and they work similar in some cases. The major difference between the two is %if-%then' can ONLY be used inside the MACRO program (whether inside the DATA step, or outside) and 'if-then' is ONLY used inside the DATA step (whether MACRO or non-MACRO).

Now, let's understand the use of Macro functions to manipulate text, perform arithmetic operations and execute SAS functions.

3. SAS Macros – Functions

These Macro functions have similar syntax, compared to their counterpart functions in data steps and they also return results in similar manner. They can manipulate macro variables and expressions and these functions get evaluated only at stage of Macro processing. Most importantly, these do not require quotes around character constant arguments.

Let's look at some of these one by one:

%UPCASE()

This function is used to convert case of letters to uppercase:

Syntax: – %UPCASE (Argument)

Example:

```
%Let heading=this is detail of country;

 Proc Print Data=Orion.Country;

 Title %UPCASE(&heading);

 Run;
```

It will print title as “THIS IS DETAIL OF COUNTRY”.

%SUBSTR()

This function returns the given number of characters from specified position.

Syntax: – %SUBSTR (argument, position [, number of characters])

If number of character is not supplied, %SUBSTR function will return characters from given position till end of the string.

Example:

```
%Let abc=this is detail of country;

 %LET def=%SUBSTR(&abc,19,7);

 Proc Print Data=Orion.Country;

 Title %Upcase(&def);
```

```
Run;
```

Here we have first extracted “country” from macro variable abc and store it into def and after that used it to show title in upper case.

%SCAN()

This function will return the nth word in a string (having n words separated by delimiters).

Syntax: – %SCAN (argument, n [, delimiter])

It will return NULL value, if string does not have n words separated by delimiter and if we have not given delimiter then it will use default value for it. Default delimiters are blank , . (& ! $ *) ; – / %.

Example: –

```
%LET STR=Analytic Vidhya.Educon;

%LET ABC=%SCAN(&STR,2);

%LET BCD=%SCAN(&STR,1,i);

%PUT &ABC;

%PUT &BCD;
```

ABC will store string “Vidhya”, here it automatically identify the second word based on default delimiter and BCD will have value “Analyt” because we have mentioned that i is a delimiter and we required the first word.

%EVAL()

This function is used to perform mathematical and logical operation with macro variables. Remember, macro variable contains only text (numerical values are also stored as text) and therefore we can’t perform any arithmetical and logical operation, if we try to use them directly. EVAL() function also returns a text result.

Syntax: – %EVAL (Expression)

Example: –

```
%LET A=3 ;

 %LET B=&A+1;

 %LET C=%EVAL(&A+1);

 %PUT &A &B &C;
```

As we know, macro variables store value as text, macro variable B will store 3+1 and with the use of %EVAL, C will store 4.

Question?

If, we create a variable like:

```
 %LET D=%EVAL (&A+3.2);
```

What would variable D store?

If you answered 6.2, you are wrong! This statement will throw up error because SAS understands "." as character. Now, to evaluate floating numbers, we will use **%SYSEVALF ()** function and above statement should be written as:

```
%LET D=%SYSEVALF(&A+3.2);
```

%SYSFUNC()

This function is used to execute SAS functions in macro environment. With the help of this function, we can increase the list of functions available to the macro language significantly. This function can allow us to make almost all DATA step and user-written functions usable in SAS Macros.

Syntax: –

%SYSFUNC (Function(arguments..) [, Format])

It has two arguments, first is for the function and second is optional to format the output of function.

Example: –

```
%Put %SYSFUNC(Today(),Date9.);
```

This will display the current date in date9 format.

%STR()

This function removes/masks the normal meaning of following token + – * /, > < = ; “ LT EQ GT LE GE LE NE AND OR NOT blank. It also preserves leading and trailing blanks of the string. Please note that this does not mask & and %.

Syntax: – %STR (argument)

%NSTR ()

This functions works exactly as %STR, but it also masks macro triggers % and &.

Example: –

```
%LET D=5;

 %LET abc = %STR ( "S &D Analytics");

 %PUT &abc;

%LET bcd = %NRSTR ( "S &D Analytics");

 %PUT &bcd;
```

Here first %PUT statement will return S 5 Analytics whereas second one will return S &D Analytics.

End Notes

This brings me to the end of this tutorial. I hope you learnt some new ways of manipulating data in SAS. In this tutorial, we looked at the basic concepts of SAS and how they become useful to accomplish repetitive tasks easily.

We also looked at how SAS Macros can be used in iterative and conditional circumstances, followed by several functions to perform text manipulations and to apply arithmetical and logical operations in SAS Macros. These, in combination with conditional statements can provide a very powerful mechanism to automate / modularize your SAS programmers.