/\*

What are Macros

Macros are used to automate the repetitive task. It can make your work faster by automating

the task that requires writing same lines of code every day. It can also be used when you

design a complex algorithm and want to make usage of code user friendly so that people who

are not comfortable with programming can use your algorithm.

Introduction to SAS Macro Programming

\* Macro Variables - A macro variable is used to store a value. The value is always character.

The character value can include variable-name, letters, numbers or any text you want

substituted in your program.

Macro Variables are of two types -

a) Local - If the macro variable is defined inside a macro code, then scope is local. It would be

available for use in that macro only and gets removed when the macro is finished.

b) Global - If the macro variable is defined outside a macro code, then scope is global. It can be

use any where in the SAS program and gets removed at the end of the session.

\*/

\* 5 ways to create macro variables ;

/\* 1. %LET

It can defined inside or outside a macro.

The syntax of the %LET statement -

%LET macro-variable-name = value;

\*/

%LET x = 5;

/\* How to use Macro Variables

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

& <Macro variable Name>

To view in log window what macro variable would return, use %PUT statement :

\*/

%put &x.;

/\* 2. Macro Parameters

Suppose you are asked to write a macro that returns mean value of a variable. The analysis variable,

input and output data sets are dynamic.

\*/

%macro test (input =, ivar=, output=);

proc means data = &input noprint;

var &ivar;

output out = &output mean= ;

run;

%mend;

\* In the above code, test is a macro, input, ivar and output are local macro variables.;

/\* 3. INTO clause in PROC SQL

Example : Calculate average height and store in a macro variable

\*/

proc means data = sashelp.heart noprint;

var height;

output out = test mean= avg\_height;

run;

Proc Print data=test;

run;

proc sql noprint;

select avg\_height into :var1 from test;

quit;

%put &var1;

\* Result : 64.81318;

/\* 4. CALL SYMPUT routine

The syntax of CALL SYMPUT :

CALL SYMPUT(macro\_varname,value);

\*/

data \_null\_;

set test;

call symput('var2',avg\_height);

run;

%put &var2;

/\* 5. ITERATIVE %DO

The syntax of iterative %DO -

%DO macro-variable = start %TO stop <%BY increment>;

. . . text . . .

%END;

\*/

%macro calcl(start,stop);

%do year = &start %to &stop;

data test;

set yr&year;

year = 2000 + &year;

run;

%end;

%mend calcl;

\* Scope of macro variable;

%let globalvar =10;

%macro abc;

%let localvar=20;

%put &localvar;

%put &globalvar;

%mend abc;

%abc;

%macro def;

%put &localvar;

%put &globalvar;

%mend def;

%def;

\* Will get warning message

WARNING: Apparent symbolic reference LOCALVAR not resolved.

a) globalvar has been defined outside of all macros, so global has global scope and all macros can access it

a) locarvar has been defined inside of macro 'abc' so it is local to macro abc and macro 'def' cannot

access it.;

\* Solution;

%let globalvar =10;

%macro abc;

%global localvar; /\* Declare scope of localvar to global \*/

%let localvar=20;

%put &localvar;

%put &globalvar;

%mend abc;

%abc;

%macro def;

%put &localvar;

%put &globalvar;

%mend def;

%def;

/\* SAS Macro Functions

There are some SAS macro functions which help you to execute various operations within a macro.

They are listed below -

1. %EVAL Function

It is used to perform mathematical and logical operation with macro variables.

\*/

%let x = 10;

%let y = 20;

%let z = &x \* &y;

%put &z;

\* It returns "10\*20";

%let z2 = %eval(&x\*&y);

%put &z2;

\* It returns 200.;

%let last = %eval(4.5+3.2);

\* returns error as it cannot perform arithmetic calculations with operands that have the floating

point values. It is when the %SYSEVALF function comes into picture.;

%let last2 = %sysevalf(4.5+3.2);

%put &last2;

\* It returns 7.7;

/\* 2. %SYSFUNC Function

There are several useful Base SAS function that are not directly available in Macro, %Sysfunc

enables those function to make them work in a macro.

\*/

%let dt3 = %sysfunc(upcase('bangalore'));

%put &dt3.;

/\* 3. %STR Function

Usage I : This function removes the normal meaning of following token

+ – \* /, > < = ; “ LT EQ GT LE GE LE NE AND OR NOT blank.

Suppose we need to store PROC PRINT; RUN; command in a macro variable.

\*/

%let exmp0 = proc print; run;;

%put &exmp0;

/\* Result : proc print

Since the semicolon following PRINT terminates the %LET statement. It does not consider RUN statement.

To workaround this issue, let's use %STR function.

\*/

%let exmpl = %str(proc print; run;) ;

%put &exmpl;

/\* Result : proc print; run; \*/

\* Usage II : Precede with % sign when you use single or double quotation in macro;

%let var=%str(a%");

%put &var;

/\* Result : a"

If you would not use %STR function in the above example, you would not be able to store quotes in

a macro variable.

\*/

\* Usage III : It also preserves leading and trailing blanks of the string.;

%let dt= %str( a );

%put &dt;

\* Run the above code and compare it with the code below, you would understand the difference ;

%let dt= a ;

%put &dt;

/\* 4. %NRSTR Function

%NRSTR works similar to %STR works except it does not resolve the % and & but stop the macro triggers.

\*/

%let exmpl = %nrstr(proc print; run;) ;

%put &exmpl;

/\* Result : proc print; run; \*/

%put "Difference between %NRSTR(&SYSDATE9) and &SYSDATE9";

\* Result : "Difference between &SYSDATE9 and 23DEC2016";

\* In the above case, %NRSTR() stops the &SYSDATE9 macro function.;

\* 5. %SCAN Function - It returns the nth word in a string.;

%let var = a b c;

%let varName =%scan(&var,1,%str( ));

%put &varName;

\* Result : a;

/\* How to store list of values in a macro variable

Suppose you have a list of names and you want to store them in a macro variable. It can be done

by using SEPARATED BY ' ' keyword in PROC SQL.

\*/

data temp;

input name $;

cards;

Rajesh

Ron

Harry

;

run;

proc sql;

select name into :myvar separated by ',' from temp;

quit;

%put &myvar; /\* Rajesh,Ron,Harry \*/

\* In this case, we have used comma(,) as a delimiter. We can use any other delimiter.;

proc sql;

select name into :myvar1-:myvar3 from temp;

quit;

%put &myvar1;

%put &myvar2;

%put &myvar3;

\* Macro Introduction - SAS macro is Used to reduce the amount of code entered to

perform common tasks. It is used for tasks that are repeated in number of programs or number of

different places within a program.;

Proc print data=mylib.abc;

run;

Proc print data=mylib.emp;

run;

Proc print data=mylib.cust;

run;

/\* Defining Macro;

%MACRO macro-name(parameter1, paramenter2,....paramentern);

macro definition

%MEND macro-name;

macro-name - is the name of the macro.

macro definition - is valid SAS step

\*/

\* Calling a Macrro

%macro-name(parameter1, paramenter2,....paramentern);

%macro mname;

Proc print data=mylib.cust;

run;

%mend mname;

%mname;

\*Passing Information into a Macro Using Parameters;

%macro macroname(datasetname);

Proc print data=Training.&datasetname;

run;

%mend macroname;

%macroname(cust);

%macroname(abc);

%macroname(emp);

Data abc;

input Custid$ Sales;

cards;

c1 200

c1 300

c1 400

c1 500

c2 2000

c2 3000

c2 4000

c2 5000

;

run;

Data def;

input Custid$ Sales;

cards;

c1 200

c1 300

c1 400

c1 500

c2 2000

c2 3000

c2 4000

c2 5000

;

run;

Proc means data=abc;

where custid='c1';

run;

Proc means data=abc;

where custid='c2';

run;

options symbolgen;

%macro secondm (dset=abc, custid='c1');

Proc means data=&dset;

where custid=&custid;

run;

%mend secondm;

%secondm;

%secondm(dset=abc);

%secondm(custid='c2');

%secondm(custid='c1',dset=abc);

%secondm(dset=def,custid='c2');

/\*Two types of parameters

1) Positional Parameters

2) Keyword Parameters

1) Positional Parameters

%MACRO macro-name(parameter1, paramenter2,....paramentern);

macro definition

%MEND macro-name;

\*/

%macro mname(datasetname,varname);

Proc print data=&datasetname;

var &varname;

run;

%mend mname;

%mname(Training.cust,ID);

/\* 2) Keyword Parameters

a) parameters followed by equal signs

b) can specify default values after the equal signs

%MACRO macro-name(parameter1=default-value1, paramenter2=default-value2,....paramentern=default-valuen);

macro definition

%MEND macro-name;

\*/

%macro mname(datasetname=Training.cust,varname=ID);

Proc print data=&datasetname;

var &varname;

run;

%mend mname;

%mname;

%mname(datasetname=Training.cust);

%mname(datasetname=Training.emp, varname=empid);

%mname(varname=empid, datasetname=Training.emp);

\* How to debug SAS Macros - There are some system options that can be used to debug SAS Macros;

/\* 1. MPRINT - MPRINT translates the macro language to regular SAS language. It displays all the

SAS statements of the resolved macro code.

\*/

options mprint;

%macro test (input =,output=);

proc means data = &input noprint;

var height;

output out = &output mean= ;

run;

%mend;

%test(input=sashelp.heart,output=test);

/\*It returns the following message in LOG window :

MPRINT(TEST): proc means data = sashelp.heart noprint;

MPRINT(TEST): var height;

MPRINT(TEST): output out = test mean= ;

MPRINT(TEST): run;

\*/

/\* 2. MLOGIC

It is very helpful when we deal with nested macros (Macro inside another macro). Often we use %DO loops and

or %IF-%THEN-%ELSE statements inside the macro code and LOGIC option will display how the macro variable

resolved each time in the LOG file as TRUE or FALSE.

\*/

options mlogic;

options mindelimiter=,;

options minoperator;

%MACRO test();

%DO i = 1 %to 9 ;

%if &i in (1,3,5,7,9) %then %do;

%PUT i = &i - odd;

%END;

%ELSE %DO;

%PUT i = &i - even;

%end;

%end;

%MEND;

%test();

/\* In the log window, see what MLOGIC option produces -

MLOGIC(TEST): %DO loop beginning; index variable I; start value is 1; stop value is 9; by value is 1.

MLOGIC(TEST): %IF condition &i in (1,3,5,7,9) is TRUE

MLOGIC(TEST): %PUT i = &i - odd

i = 1 - odd

MLOGIC(TEST): %DO loop index variable I is now 2; loop will iterate again.

MLOGIC(TEST): %IF condition &i in (1,3,5,7,9) is FALSE

MLOGIC(TEST): %PUT i = &i - even

i = 2 - even

\*/

\* 3. SYMBOLGEN - It writes the results of resolving macro variable references to the SAS log for debugging.;

options symbolgen;

%macro test (input =,output=);

proc means data = &input noprint;

var height;

output out = &output mean= ;

run;

%mend;

%test(input=sashelp.heart,output=test);

\* SYMBOLGEN: Macro variable INPUT resolves to sashelp.heart

SYMBOLGEN: Macro variable OUTPUT resolves to test;

/\* Difference between MPRINT and SYMBOLGEN

See the log generated by these two options. MPRINT option prints all the statements within the macro

(not just macro variables). Whereas, SYMBOLGEN option prints only the results of macro variables.

\*/

\* How to turn off macro debugging options;

options nomprint nomlogic nosymbolgen;

\* All of these options can be turned off together as specified above. Or one of them can be turned off

and remaining ones keep turned on.;

\* Use Proc PRINTTO for saving log in an external text file.;

proc printto log="/home/u1048896/LOG2.txt" new;

run;

\* Clear LOG and OUTPUT Window;

DM "Log" clear continue;

DM "Output" clear continue;