SAS Macros

Module Content

- Uses of Macros
- Global and Local Macro Variables
- Creating macros using %LET
- Creating Macros using PROC SQL
- Call Symput
- Iterative statements
- Conditional Processing
- SAS Macro functions
- Inbuilt macro variables
- Debugging

Uses of Macros

- Creating a macro using a section of code allows this code to be repeated over and over with the ability to change values in each run.
- It makes the complex programs simple and sometimes even efficient.
- By setting a value as a macro variable it reduces the risk of errors should this value change as it is only defined once, or enables this value to vary if defined in a call symput statement.
- conditionally execute DATA or PROC steps

Global and Local Macro Variables

Global macro variables are defines outside of program and available for till the session is running. They get removed once session is closed.

Local Macro variable are declared as part of the program and mostly used to supply different variables to the same SAS statements so that they can perform a process in a data set. They get removed when macro is finished.

Creating Macro Variables and Resolving it

%let statement

- %let test=***one***;
- test is the name of the macro variable
- one is the value of the macro variable test
- value assigned can contain letters, numbers, printable characters and blanks

After a macro variable is created you can reference the variable as follows:

- &test
- %put &test;
- returns the value ***one***

Exceptions with %let

- **Acceptable**

```
%let a = abc;
%let &a = pqr;
%put &abc;
```

- **Not Acceptable**

```
%let a = 13;
%let &a = pqr;
%put &13;
```

Acceptable – Log:

```
207          %let a = abc;
208        %let &a = pqr;
209        %put &abc;
pqr
```

Not Acceptable – Log:

```
204  %let a = 13;
205        %let &a = pqr;
ERROR: Expecting a variable
name after %LET.
206        %put &13;
&13
```

Macro variables Multiple Ampersands

```
%let name  = grand;
%let grand = Oberoi;
%let Oberoi= hotel;
```

Creating Macro Variables

```
%put &name;
```

&name > grand

```
%put &&&name;
```

&& &name > & grand > Oberoi

```
%put &&&&&&&name;
```

&& && && &name > & & & grand >

&& &grand > &Oberoi > hotel

Resolving Macro Variables

Define – Compile - Invoke

- A macro can be defined at any point in a SAS program
- Syntax – Begin definition with

  ```
  %macro <name>;
  ```

  End macro with

  ```
  %mend <name>;
  ```

  Putting the macro name on the %mend statement is not necessary but is good practice.
- Run the code beginning with %macro ending in %mend to compile the macro. SAS stores the macro and the code does not need to be run again unless it is changed
- Invoke the macro by calling it to run

  Syntax - %<name>

Delimiters triggering macro processor activity

- &name

```
%let myname = Gourab;
%put my name is &myname;
```

- %macro

```
%macro myname;
Gourab
%mend myname;
%put My Name is %myname;
```

Inserting Comments in Macros

- /*   ---- COMMENTS---   */
- %* ----COMMENTS---   ;

Macro Definition Containing several SAS statements

```
%macro print_var(var1=);

proc print data=city_info;

var &var1;

run;

%mend print_var;
```

Passing Information in a Macro

```
%print_var(var1=name);

%print_var(var1=city);
```

Positional and Keyword

- For example, a macro called test with variables month and dset
  - 4 Using positionally defined parameters

```
%macro test(month,dset);
        <Bits of code>
%mend;
%test(june,loans);
```

  - 4 Using keyword parameters

```
%macro test(month=,dset=);
        <Bits of code>
%mend;
%test(month=june,dset=loans);
```

- It is good practice to use keyword parameters to avoid confusion
- Combination of both is possible – positional parameters should precede keyword parameters in both macro definition & macro call

When Quote? When not?

- %let var1 = city;
- %let var2 = age;
- Compare two variables
  - If &var1 eq &var2;
- Compare a character variable with a value
  - If &var1 eq “&var2”;
- Compare two character values
  - If “&var1” eq “&var2”;

Macro variable separator

```
%let client1= GAP;
%let client2 = OLD NAVY;
%put &client1.and&client2;
```

The period operator separates a macro variable from a normal text

Macro variable/File separator

```
%let filename = text_data;
%let full_file_name = &filename..xls;
%put &full_file_name;
```

Examples

- Simple SAS code

```
data loans_arrears;
set perform.perform0704(keep= prodid prod_mkr currstat);
where prod_mkr='LP' and currstat ge '03';
run;
data ca_arrears;
set perform.perform0704(keep= prodid prod_mkr currstat);
where prod_mkr='AA' and currstat ge '03';
run;
```

- How this could be made a macro

```
%macro arrears(dset=,prod=);
data &dset._arrears;
set perform.perform0704(keep= prodid prod_mkr currstat);
where prod_mkr=&prod. and currstat ge        '03';
run;
%mend arrears;
%arrears(dset=loans,prod='LP');
%arrears(dset=ca,prod='AA');
```

Creating Macros using PROC SQL

- proc means data = sashelp.heart noprint;
  var height;
  output out = out1 mean= mean_ht;
  run;

- proc sql noprint;
  select mean_ht **into :**var1
  from out1;
  quit;
  %put &var1;

CALL SYMPUT routine

A specific value can be assigned within the data step to a macro variable using Call Symput statement

```
data _null_;
set test;
call symput ('var2',avg_height);
run;

%put &var2;
```

Iterative Macro Statements

%DO statement executes a portion of a macro multiple times as per the value of an index variable.

```
%macro create(No);
    %do i=1 %to &No;
        data month&i;
        infile test&i;
        input name $ age dob $;
        run;
    %end;
%mend create;

%create(2)
```

Resolves to:

```
DATA MONTH1;
INFILE test1;
INPUT NAME $ age dob$;
RUN;
DATA MONTH2;
INFILE test2;
INPUT name $ age dob$;
RUN;
```

This will input multiple files with names starting with test eg test, test2

Conditional Processing

%IF-%THEN/%ELSE Macro Statement conditionally process a portion of a macro

```
%MACRO create();
        %DO i = 1 %to 10 ;
                %if &i in (2,4,6,8,10) %then %do;
                %PUT i = &i - even;
        %END;
        %ELSE %DO;
                %PUT i = &i - odd;
        %end;
        %end;
%MEND;


%create();
```

SAS Macro Functions

%UPCASE()

- This function can be used to transform case of letters to uppercase letters.

```
%Let title= name of doc;
Proc Print
            Data=test;
Title %UPCASE(&title);
```

SAS Macro Functions

%SUBSTR()

- returns the number of characters from mentioned position

```
%Let Title=name of document;
%LET var=%SUBSTR(&title,5);
Proc Print Data=sashelp.cars;
        Title %Upcase(&var);
Run;
```

SAS Macro Functions

%SCAN()

- This function will return the specified nth word in a provided string.

```
%LET STR=test;
 %LET STR2=%SCAN(&STR,2);
 %LET STR3=%SCAN(&STR1,1,a);
 %PUT &STR2;
 %PUT &STR3;
```

SAS Macro Functions

- **%EVAL()**

mathematical and logical operation can be performed over macro variables using this function

```
%LET Year1=2001;
 %LET Year2=&A+9;
 %LET Year3=%EVAL(&A+10);
 %PUT &Year1 &Year2 &Year3;
```

SAS Macro Functions

%SYSFUNC()

- SAS functions within macro environment can be executed using this function.

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

%STR() and %NSTR

- the normal meaning of following token can be masked using these functions

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

%LET test = %STR (“A &K Enterprises");

%LET K=BC;

%LET test = %NRSTR (" A &K Enterprises ");

In-Built Macro Variables

The SAS macro processor creates automatic macro variables whenever SAS is started that stores information related to the SAS session.

&SYSTIME

the time a SAS session or job started executing

&SYSLAST

most recent SAS data set name

&SYSDATE

the date a SAS session started executing (YY – year format)

SYSDATE9

the date a SAS session started executing (YYYY – year format)

Debugging

- There are options which can be switched on and off which make it easier to see why a macro might not be working.
  - Mprint – prints all the code created by the macro
  - Mlogic – prints the logic the macro uses e.g. whether conditions in %if-%then-%else clauses are met
  - Symbolgen – shows the resolved values of all macro variables used
  - Use nomprint nomlogic and nosymbolgen to switch them off
- %put _user_; lists all user defined macro variables in the log window
  - Others include
    - %put _all_;
    - %put _global_;
    - %put _local_;
    - %put _automatic_;
- Make sure statements match such as %MACRO-%
- Be careful about balancing single and double quotes.
- Be careful to spell all keywords including macro names correctly.

greatlearning
Learning for Life

Thank you