sas macros

Juan Shishido, D-Lab, UC Berkeley

Month Day, 2016

introduction

macro facility

The SAS macro facility enables the generation of text in SAS programs

It has two components:

- macro language—is text based and defines the syntax needed for writing macro statements
- macro processor—resolves macro statements into standard SAS statements

macro variables and macros

The macro processor is activated when either & or % characters are encountered

These refer to macro variables and macros, respectively

macro variables

Macro variables are used to represent text and:

- can be defined and used anywhere in SAS programs
- do not depend on SAS data sets
- correspond to a single value (until explicitly changed)

They are used to substitute text in SAS programs

macros

Also enable the substitution of text in SAS programs, but can include:

- macro variables
- DATA and PROC steps
- other macro statements

scope

SAS programs can include any number of macro variables or macros

Macro variables defined outside of macros are considered global, which means they can be referenced anywhere in the program

When they are defined inside of macros, however, they are local and can only be referenced within the macro

It is possible, as we will see, to create global macro variables inside of macros by using the `%global` statement

use cases

Macro variables and macros are useful in situations where

- code is repetitive
- values must be dynamic
- execution of code depends on other values

syntax

defining macro variables

The easiest way to define a macro variable is by using the `%let` macro statement

For example

```
%let workshop = SAS;
```

Notice that quotes are *not* needed

In a program, we would reference this using `&workshop`

using macro variables

Suppose that the D-Lab needed to print details for a given set of workshops

The print procedure might look like

```
proc print data = SAS;
    var = description;
    title 'D-Lab Workshop on SAS';
run;
```

For a particular SAS workshop

using macro variables

Suppose the D-Lab also wanted needed to print descriptions for its Python and R workshops

Rather than repeating the print statements, we could use the `&workshop` variable as follows

```
%let workshop = Python;
proc print data = &workshop;
    var = description;
    title "D-Lab Workshop on &workshop";
run;
```

using macro variables

And for R

```
%let workshop = R;
proc print data = &workshop;
    var = description;
    title "D-Lab Workshop on &workshop";
run;
```

Notice that we only have to replace the value of `&workshop` to get the descriptions for the other variables

Also, when calling macro variables that are part of strings, we must use double quotes

periods

In some cases, macro variables may only be part of a needed string

Let's say we had the following SAS data sets: `us_gdp`, `ca_gdp`, `us_pop`, and `ca_pop`

We could reference each of these four data sets individually

Because we know about macro variables, however, we could create two references

periods

We could do something like the following

```
%let country = us;
%let metric = gdp;
```

To reference `us_gdp`, we would do the following

```
&country._&metric
```

Notice the period (`.`) that appears before the underscore (`_`)

periods

Periods are used as delimiters that define the end of a macro variable

Had we not included the period, SAS would have interpreted the first macro variable reference as `&country_`, which does not exist

If the data set names did not include underscores, we could have simply used `&country&metric` because the ampersand (`&`) lets SAS know that its encountered a new macro variable

periods

In cases where a period is actually part of the name, we use double periods

```
%let filename = dlab;

data &filename;
    infile &filename..txt;
    input ...;
run;
```

The macro variable in the `infile` statement resolves to `dlab.txt`

multiple ampersands

In other cases, two macro variables might be used to reference a third

```
%let bldg = barrows;
%let room = 350;
%let barrows350 = dlab;
```

`&bldg&room` resolves to the value `barrows350`

SAS resolves `&bldg` and then `&room`, concatenating those values

multiple ampersands

Because we also defined a macro variable with that name, we can use multiple ampersands to access its value

`&&&bldg&room` resolves to `&barrows350` which resolves to `dlab`

Here is what the macro looks like during each scan:

1. `&&bldg350`
2. `&barrows350`
3. `dlab`

creating macro variables from data steps

We can also create macro variables using the `call symput` routine

The syntax is as follows: `call symput(macro-variable, value)`

call symput

```
data _null_;
    input campus $ enrollment;
    call symput(campus, trim(left(enrollment)));
datalines;
berkeley 37581
davis 35415
la 43239
;
run;
```

This creates three macro variables

%put

We can use the macro variables `&berkeley`, `&davis`, and `&la` anywhere in the SAS program in which they were created

A useful way to write text to a `.log` file is by using `%put`

```
%put "The enrollment at Berkeley in 2014 was &berkeley.."
```

macros

So far, we've worked with macro variables

Now, let's explore macros

macros

```
%macro macro-name <(macro parameters)>;
    macro text
%mend <macro-name>;
```

Macro definitions always start with the `%macro` keyword followed by a macro name

Macro parameters, which are optional, should be separated by commas

Macro definitions always end with the `%mend` keyword, though the macro name is optional

parameters

other macro statements

references

links

- SAS Macro Programming for Beginners
- SAS 9.3 Macro Language Reference
- Get Started Writing SAS Macros
- Get Started with Macro
- Resolving and Using &&var&i Macro Variables