Introduction to SAS macros

http://www2.sas.com/proceedings/sugi29/243-29.pdf

Learning objectives

In this shortened section, we will introduce ourselves to the SAS macro functionality.
We will see three ways that macros can extend the capabilities of SAS

1. Use of macro variables to modify SAS code
2. Use of macros to make modular SAS code
3. Use of macros to make SAS code that does different things depending on data input

What are SAS macros?

- When you submit SAS code for processing, SAS compiles it and executes it.
- If your code includes macro statements, these statements are resolved first, creating vanilla SAS code. Then the SAS code (with macros resolved) is compiled and executed as usual.
- SAS macros are indicated by %
  - Can contain DATA and PROC steps
  - Can also contain macro statements like %IF-%THEN-%ELSE and %DO-%END
  - Can also contain macro variables
- SAS macro variables are indicated by &
  - Like an ordinary data variable except must be character type and doesn't belong to a data set
  - Can be used to store things like
    - variable names
    - numerals
    - text strings to be plugged in to your SAS program

Macro variables

- Macro variables defined inside a macro have *local scope*
- Macro variables defined outside a macro have *global scope*
- The macro processor won't find macro variables inside single quotes - use double quotes instead
- To assign a value to a macro variable, use %LET
  - e.g. `%LET macroVariableName = value;`

Example: Define the macro variable `winner` and assign it a value:
`%LET winner = Lance Armstrong;`
Now, if we want to use this macro variable, for example in a title statement, we use the & so that the SAS macro processor will identify the macro variable:

```
TITLE "First:  &winner";
```

When the macro processor goes through the SAS program, it will substitute the value of the macro variable, like so:

```
TITLE "First:  Lance Armstrong";
```

Modular code with macros

- If you have a block of code that you use frequently, you can define a macro and place the macro name in your SAS program instead.
- Macro syntax is:
  `%MACRO macro-name;`
  `macro-text`
  `%MEND macro-name;`
- Once you have defined your macro, you can insert it into your SAS program with
  `%macro-name`

Modular code with macros

- You can add parameters to macros, similar to function calls in programming languages. The parameters become local macro variables within the macro.

```
%MACRO macro-name (parameter-1=, parameter-2=,
...  parameter-n=);
macro-text
%MEND macro-name;
```

- You can then (optionally) supply values to the parameters when you insert the macro.

```
%macro-name (parameter-1=value-1, parameter-2=);
```

Conditional logic

- You can use conditional logic for flow control of macros

```
%IF condition %THEN action;
%ELSE %IF condition %THEN action;
%ELSE action;

%IF condition %THEN %DO;
action;
%END;
```

- This makes macros much more flexible, and able to generate different SAS code depending on input conditions

Data-driven programs

- CALL SYMPUT is a macro routine that allows a macro program to peek at the data
  - It takes a value from a DATA step and assigns it to a macro variable
  - You can't use CALL SYMPUT to generate a macro variable and use that variable in the same data step
    - The data step must execute in order to get the value. CALL SYMPUT then passes the created macro variable back to the macro processor.
- Syntax:

```
CALL SYMPUT("macro-variable", value);

IF Place = 1 THEN CALL SYMPUT("WinningTime",
Time);
```

  - When `Place` is equal to 1, the macro variable `&WinningTime` will be assigned the value `Time`

Seeing what SAS sees

Debugging macros can be difficult.

- You can insert print statements to try to figure out where bugs live
- You can use the MPRINT option
  `OPTIONS MPRINT;`
  - This will print out the resolved macro statements so that you can see what the SAS program ends up looking like before compiling and execution.
  - The SAS program output will go to the log.
- Another suggestion is to create a vanilla version of the SAS program and ensure that it works as intended before modifying it into a macro.
- Make incremental changes and test at each change.