CODE DESIGN, MACROS, AND SUBROUTINES

CODE DESIGN

- Do not simply start writing code!!! Attempt to write your best version first.
- Develop a detailed project outline.
- Choose the best tool for the job (SAS is great for data prep and large datasets, while Stata works well for estimating models)
- Break your project into discrete steps that can be developed independently.
  - Create input data (subset, sort, and merge input files)
  - Transform (create new variables)
  - Statistical estimates (develop estimators, generate models)
  - Output and diagnostics (create tables, graphs, and verify results)

ASSUME YOUR CODE IS WRONG!

- Design your code and subroutines under the assumption you will make mistakes
  - During development, break your program into small pieces and debug each section of code separately
  - Add in debugging code that will be turned off during normal execution (i.e. put statements in SAS)
  - Develop a prior expectation of what the results will look like at each stage

CODE STYLE

- Make your code as transparent as possible
  - Other users and even yourself in a few months will forget important details that seem clear while writing the code
- Use a consistent style
  - Indent code blocks
  - Create useful comments (not as easy as it seems)
  - Use lower or upper case consistently. Do not mix unless the style benefits are very important (preserve search ability)
- Sequentially number your code sequence and tag datasets (runall.bash)

EXAMPLE

```
libname dot “.”;
data temp1;
set dot.inputs;
if good=1;
proc means;
proc reg;
model y=x1 x2 x3;
```

REWRITE 1

```
*** Use the data created in program 0_inputs.sas  ***;
*** to calculate the returns to experience        ***;

libname dot ".";

data temp1;
   set dot.inputs;
   if good=1 then output;
run;

proc means data=temp1;
run;

proc reg data=temp1;
   model y=x1 x2 x3;
run;
```

REWRITE 2

```
*** Use the data created in program 0_inputs.sas  ***;
*** to calculate the returns to experience        ***;

libname dot ".";

proc means data=dot.inputs(where=(good=1));
run;

proc reg data=dot.inputs(where=(good=1));
   model y=x1 x2 x3;
run;
```

REWRITE 3

```
*** Use the data created in program 0_inputs.sas  ***;
*** to calculate the returns to experience        ***;

libname dot ".";

/* Add a quartic experience term */

data temp1 /view=temp1;
   set dot.inputs(where=(good=1));
   length x4 5;
   x4=x3*x;
run;

proc means data=dot.inputs data=temp1;
run;

proc reg data=dot.inputs(data=temp1);
   model y=x1 x2 x3 x4;
run;
```

MACROS

- Enables you to delay resolution of a portion of your code.
- You can replace a hard-coded filename, variable, or section with dynamic code that is generated at the time of execution
- SAS and Stata process macros similarly, but SAS passes your entire program through the macro pre-processor, while Stata operates line by line.
  - Result: If there is an error, SAS will fail prior to processing data while Stata will run until the first macro failure
  - Stata uses macros in built in commands (ado files). Using global macros can have unintended effects. You can unknowingly overwrite an existing variable or use a macro variable created in another command

SIMPLE EXAMPLE

- SAS
  - %let rhs_vars=x1 x2 x3;
  - proc reg; model y=&rhs;
- Stata
  - local rhs_vars=x1 x2 x3;
  - reg y `rhs_vars'

SAS MACRO FACILITY

- Macros can be used to both generalize a program and create subroutines
- By processing is often a more transparent solution (available in both SAS and Stata). Similar set of processing for multiple groups (states or industries for example).
- Always use MPRINT, MLOGIC, and SYMBOLGEN options
- %include (great for config files) and autocall (sasautos)
- When debugging a difficult section of code, output the resolved code to a file
- grep MPRINT
- Exercise. Run code and output the resolved code to a file. Run that code separately and look at the results.
- See: http://www2.sas.com/proceedings/sugi29/243-29.pdf

WHY USE SUBROUTINES?

- Using a code library improves the productivity and quality of your code
  - Encourages the re-use of **well tested** algorithms
  - Enforces consistent results across time and team members
- Some loss of transparency
  - Code in subroutine is hidden from view when called from the main program

SUBROUTINES

- Subroutines can be isolated from the main code
- Provide a local name space
- Returning results to the main program

EXAMPLE: CPI MACRO

```
*** Adjust earnings for inflation ***;

libname dot ".";

%macro cpi;
   array cpi{2010:2013} _TEMPORARY_;
   if _N_=1 then do;
      cpi{2010}=.8;
      cpi{2011}=.9;
      cpi{2012}=.95;
      cpi{2013}=1.2;
   end;
%mend;

data temp1;
   set dot.inputs;
   %cpi;
   earn_adj=earn*(cpi{2010}/cpi{year});
run;
```

BAD EXAMPLE: CPI

```
*** Adjust earnings for inflation ***;

libname dot ".";

%macro cpi;
   data temp1;
      set dot.inputs;
      array cpi{2010:2013} _TEMPORARY_;
      if _N_=1 then do;
         cpi{2010}=.8;
         cpi{2011}=.9;
         cpi{2012}=.95;
         cpi{2013}=1.2;
      end;
      earn_adj=earn*(cpi{2010}/cpi{year});
run;
%mend;
```

STATA

- Links to web tutorials
  - http://www.ssc.wisc.edu/sscc/pubs/stata_prog1.htm
  - http://data.princeton.edu/stata/programming.html
- Ado files - Use a local ado folder and your programs can be used like any other Stata command

MATRIX PROGRAMMING

- SAS IML
- Stata Mata
- Matlab (the most efficient solution)
  - Much of the syntax is similar to Fortran
  - Useful tool for developing and prototyping Fortran solutions

CONCLUSION

- Write code with a consistent, readable style
- Use macro loops as the last tool to solve a problem
  - If you do use macros, minimize their impact by using them like subroutines
- Subroutines are a useful way to impose consistency
- Choose the best tool for the job