The coding conventions described here were adopted by the authors and their colleagues at ORI, Inc. for pragmatic rather than esthetic reasons. Though not always necessary or desirable, the use of these conventions generally saved more time and effort than was needed to implement them.

Each convention helped in at least one of the following areas:

I. Maintenance - Made it easier to modify code after the initial testing and debugging.

II. Documentation - Clarified the purpose of the program, and of its component parts. Explained less obvious coding algorithms and made code more readable.

III. Efficiency - Helped minimize the computer resources needed to execute the program.

IV. Error Prevention and Debugging - Helped the programmer avoid logical and syntactical errors, or made it simpler to trace the causes of such errors.

The coding conventions are grouped by the major reason for adoption.

I. Maintenance

A. When using a conditional DO loop (indexed DO, DO WHILE, or DO UNTIL) after a THEN or ELSE statement, nest it within an unconditional DO loop.

Not Recommended:

IF X = 4 THEN
   DO I = 1 TO 5; /* INDEXED LOOP */
      other SAS statements
   END; /* INDEXED LOOP */

Recommended:

IF X = 4 THEN
   DO; /* UNCONDITIONAL LOOP */
      SAS statements can be added here
      DO I = 1 TO 5; /* INDEXED LOOP */
      other SAS statements
   END; /* INDEXED LOOP */
   END; /* UNCONDITIONAL LOOP */

B. If a DATA step, uses the same constant frequently, substitute a variable initialized in a RETAIN statement. Then the constant value can be changed with only one modification, instead of many. Efficiency will also be enhanced.

Not Recommended:

DATA OKAY;
   other SAS statements
   IF ( A LT 1 ) THEN
      A = 1;
      IF ( 1 LE B LE 10 ) THEN
         DO;
            B = 1;
            C = 1;
         END;
   END;

Recommended:

DATA BETTER;
   RETAIN ONE 1 TEN 10;
   other SAS statements
   IF ( A LT ONE ) THEN
      A = ONE;
      IF ( ONE LE B LE TEN ) THEN
         DO;
            B = ONE;
            C = TEN;
         END;
   END;

C. When multiple data sets are created in a single data step, use PL/I-style comments to describe each data set. This convention is also helpful for documentation. Remember that in the IBM-DS environment, PL/I-style comments should begin after the first column, or execution will stop immediately.

Recommended:

DATA WATER /* PIPED-IN WATER, ONLY */
   SEWER /* SEWER CONNECTION, ONLY */
   BOTH; /* PIPED-IN WATER AND SEWER CONNECTION */
D. When statements are used to generate diagnostic printout, they should be clearly identified with the PL/I-style comment, /* TEST */, at the end of each line. When the diagnostic printout is no longer needed, convert these statements to SAS comments by inserting an asterisk before each statement. Then, when retesting is needed after program modification, the test printout can be reactivated by deleting the leading asterisks.

Recommended:

```
*PROC PRINT DATA = WOMEN (OBS = 50); /* TEST */
*TITLE TEST PRINT FIRST 50 FEMALES;/* TEST */
```

II. Documentation

A. Begin all ARRAY names with a single underscore ("_"). Then array names can immediately be distinguished from variable names.

Recommended:

```
DATA ADD_TAX;
ARRAY _"PRICE (1) ITEM1 ITEM2;
SET INVENTORY;
DO J = 1 TO 5;
    _"PRICE = PRICE * .05;
END;
```

B. Put only one source code statement per line for better legibility.

Not Recommended:

```
DATA MESSY; SET OTHER; IF A = 10; PROC FREQ
DATA = MESSY; TABLES A*B C*D; PROC PRINT DATA = MESSY; TITLE VERY UNTIDY;
```

Recommended:

```
DATA BETTER;
SET OTHER;
IF A = 10;
PROC FREQ DATA = BETTER;
TABLES A*B C*D;
PROC PRINT DATA = BETTER;
TITLE MUCH NEATER;
```

C. Line up all DO statements with their corresponding END statements and put identical PL/I-style comments on the same line. This makes the code far clearer, especially in the case of nested DO loops, and also makes maintenance and debugging far simpler.

Not Recommended:

```
IF X = 1 THEN
    DO;
        other SAS statements
        DO I = 1 TO 5;
            other SAS statements
        DO UNTIL ( J = K );
            other SAS statements
        END;
    END;
END;
```

Recommended:

```
IF X = 1 THEN
    DO; /* OUTER LOOP */
        other SAS statements
        DO I = 1 TO 5; /* MIDDLE LOOP */
            other SAS statements
        DO UNTIL ( J = K ); /* INNER LOOP */
            other SAS statements
        END; /* INNER LOOP */
    END; /* MIDDLE LOOP */
    END; /* OUTER LOOP */
```

D. All non-executable statements in a DATA step should follow the DATA statement and should precede all executable statements. The first non-executable statement should be the LENGTH statement, if it is used.

Recommended:

```
DATA NEW;
LENGTH A $3;
RETAIN B C;
ARRAY LMENT (K) ELMT1 - ELMT4;
KEEP A'B'C;
SET OLD;
```

Executable SAS statements
E. Use a standard indentation scheme to make code more readable. All PROC or DATA statements should begin in column one. Any other statements should be indented by one unit (at least 2 spaces), or more. In addition, the SAS statement following THEN or ELSE should be on the next line and indented one unit more, and all statements between DO and END statements should also be indented one unit with respect to the DO statement.

Recommended:

```
DATA SAMPLE;
  SET OLD;
  IF Z = 2 THEN
    DO; /* NOTE INDENTATION */
      other SAS statements
    END; /* NOTE INDENTATION */
  other SAS statements
```

F. Use meaningful variable and data set names, if possible.

Not Recommended:

```
DATA A;
  SET B;
  VI = DIM1 * DIM2;,
  V2 = VI * DIM3;
RUN;
```

Recommended:

```
DATA MEASURE;
  SET DIMENSNS;
  AREA = WIDTH * BREADTH;
  VOLUME = AREA * HEIGHT;
RUN;
```

G. When using the END = option on a SET, MERGE, or UPDATE statement, set END = LASTREC, to emphasize that the variable has a value of 1 only for the final record.

Recommended:

```
DATA NEW;
  SET OLD END = LASTREC;
  other SAS statements
  IF LASTREC THEN
    PUT 'GRAND FINALE';
RUN;
```

H. To delimit and clarify DATA or PROC steps, use a RUN statement after the final executable statement, and at least one blank line between steps. If a comment is needed to explain the step, it should appear either immediately before or immediately after the DATA or PROC statement. The RUN statement will force any notes generated during execution to be printed on the log within the step which generated them.

Recommended:

```
*THIS STEP COMPUTES TAXES AND MARKUPS;
DATA PRICES;
  SET COSTS;
  additional SAS statements
RUN;
```

I. To delimit major (multi-step) blocks of code, they should be preceded by PAGE statements, which will force the code to begin on a fresh page of the log, and prefaced by additional comments. The beginning and end of each block of code should be clearly identified.

Recommended:

```
PAGE;
*BEGIN EDITCHEK MODULE;
******************************************************************************
* SECTION 3 ;
* THIS SECTION WILL RUN EDIT CHECKS ON;
* ALL VARIABLES IN THE UPDATE COMMANDS;
******************************************************************************;
DATA ONE;
  other SAS statements
DATA TWO;
  other SAS statements
PROC PRINT DATA = TWO;
*END EDITCHEK MODULE;
```

J. MACROS should be defined before any unconditionally executed code, if they consist of one or more complete statements.
III. Efficiency

A. Use SUM statements rather than RETAIN statements with assignment statements to accumulate totals.

Not Recommended:
RETAIN TOTAL 0;
other SAS statements
TOTAL = TOTAL + COST;

Recommended:
TOTAL + COST;

B. In an ARRAY statement the index variable should be defined explicitly, and in processing the array within a DO loop, the indexed DO loop should be used in preference to the DO OVER "array name" loop. The DO OVER loop should only be used when the number of elements in the array is undefined.

Not Recommended:
ARRAY _VARLST VARLST1 - VARLST10;
other SAS statements
DO OVER _VARLST; /* NO EXPLICIT INDEXING */
other SAS statements
END; /* NO EXPLICIT INDEXING */

Recommended:
ARRAY _VARLST (J) VARLST1 - VARLST10;
other SAS statements
DO J = 1 TO 10; /* INDEXED LOOP */
other SAS statements
END; /* INDEXED LOOP */

IV. Error Prevention and Debugging

A. For table lookups or recoding, use a PROC FORMAT with a PUT function if there are more than 2-10 possible codes. It is much easier to change the values in a format which will serve an entire program than to modify code resident in each DATA step of a large program.

Not Recommended:
DATA NEW;
SET OLD;
IF ( 0 LE AGE LE 11 ) THEN
GROUP = '1';
ELSE IF ( AGE LT 21 ) THEN
GROUP = '2';
ELSE IF ( AGE LT 45 ) THEN
GROUP = '3';
ELSE
GROUP = '4';
RUN;

Recommended:
PROC FORMAT;
VALUE AGEGRP
0 - 10 = 1
11 - 20 = 2
21 - 44 = 3
OTHER = 4;
RUN;
DATA NEW;
SET OLD;
GROUP = PUT ( AGE, AGEGRP. );
RUN;

B. For nested IF - THEN - ELSE IF, arrange the conditions in order from most likely to least likely. Then, in most cases, fewer statements will need to be executed.

Not Recommended:
IF BEST_PET = 'SNAKE' THEN
SAS statement
ELSE IF BEST_PET = 'GERBIL' THEN
SAS statement
ELSE IF BEST_PET = 'CAT' THEN
SAS statement

Recommended:
IF BEST_PET = 'SNAKE' THEN
SAS statement
ELSE IF BEST_PET = 'GERBIL' THEN
SAS statement
ELSE IF BEST_PET = 'CAT' THEN
SAS statement
Recommeded:

\[
\text{IF BEST\_PET = 'CAT' THEN}
\]
\[
\text{SAS statement}
\]
\[
\text{ELSE}
\]
\[
\text{IF BEST\_PET = 'GERBIL' THEN}
\]
\[
\text{SAS statement}
\]
\[
\text{ELSE}
\]
\[
\text{IF BEST\_PET = 'SNAKE' THEN}
\]
\[
\text{SAS statement}
\]

C. Any statements which use values of the FIRST, LAST, IN = or END = variables must be executed before any DELETE, RETURN or subsetting IF statements.

Not Recommended:

DATA ENROLLED;

\[
\text{SET STUDENTS END = LASTREC;}
\]
\[
\text{IF CREDIT GT 0; /* SUBSETTING IF */}
\]
\[
\text{other SAS statements}
\]
\[
\text{IF LASTREC THEN}
\]
\[
\text{OUTPUT ENROLLED;}
\]
\[
\text{RUN;}
\]
\[
*\text{IF LAST RECORD HAS CREDIT = 0, NO RECORD WILL BE OUTPUT;}
\]

Recommended:

DATA ENROLLED;

\[
\text{IF LASTREC THEN}
\]
\[
\text{OUTPUT ENROLLED;}
\]
\[
\text{SET STUDENTS END = LASTREC;}
\]
\[
\text{IF CREDIT GT 0; /* SUBSETTING IF */}
\]
\[
\text{other SAS statements}
\]
\[
\text{RUN;}
\]

E. When using the SASB2 MACROs, system options MPRINT, SYMBOLGEN and MACROGEN should be used during testing and debugging. The MACROS will then be expanded and the MACRO variables will be fully resolved, so that SAS-generated notes and error messages will be meaningful. Example:

\[
\text{OPTIONS MPRINT SYMBOLGEN MACROGEN;}
\]

To summarize, the following coding conventions have been recommended and described:

- Nest indexed DO loops which follow an IF or ELSE statement within unconditional DO ... END loops.
- Replace frequently used constants in a DATA step with variables initialized in a RETAIN statement.
- Use PL/I-style comments to describe each data set when multiple data sets are created.
- Use PL/I-style comments to identify code which is used for testing, and leading asterisks to "comment out" such code when it is no longer needed.
- Begin all array names with a single underscore (\_).
- Put only one statement of source code per line.
o Use formats with PUT functions for table lookups and recoding, instead of multiple conditional assignment statements.

o When using nested IF ... THEN ... ELSE IF statements, check for the most likely conditions first.

o Always identify input or output data sets explicitly by name.

o Use subsetting IF, DELETE or RETURN statements with care that they do not precede statements which should be executed for all observations.

o When testing and debugging SAS82 MACROS, use the MPRINT, SYMBOLGEN and MACROGEN options.

The author can be contacted at:

ORI, Inc.
Software Applications and Training Program Office
7910 Woodmont Ave., Suite 1405
Bethesda, MD. 20814
(301) 656-3276