Paper 11667-2016

Can you decipher the code? If you can, maybe you can break it.

Jay Iyengar, Data Systems Consultants LLC, Oak Brook, IL

ABSTRACT

You would think that training as a code breaker, similar to those employed during the Second World War, wouldn't be necessary to perform routine SAS® programming duties of your job, such as debugging code. However, if the author of the code doesn't incorporate good elements of style in their program, the task of reviewing code becomes arduous and tedious for others. Style touches upon very specific aspects of writing code; indentation for code and comments, case for keywords, variables, and data set names, and spacing between PROC and DATA steps. Paying attention to these specific issues will enhance the reusability and lifespan of your code. By using style to make your code readable, you'll impress your superiors and grow as a programmer.

INTRODUCTION

SAS programmers are concerned with producing results. Whether it's a report, or a new data set, they are focused on shaping data to conform to a result. From this focus on results naturally evolve two specific issues to pay attention to when writing code.

1. Is your code free of errors, warnings, and notes that highlight potential problems?
2. Have you selected the right SAS tool or method to produce the intended result?

The first issue is quite straightforward. The second issue I've mentioned is really based on the project or task specifications/instructions that you're given. For a given task, you might ask yourself if you should use a PROC SQL INNER JOIN or OUTER JOIN to combine data sets, or instead a DATA STEP Match-Merge.

There's a third specific issue which affects results, and a programmer's ability to produce results. Its style. In some work environments, style receives substantially less attention than the first two issues. Each programmer has their own individual style. Style impacts the readability and thus the reusability of code. Style thus plays a role in work efficiency and project management.

I recently worked on a project where I was asked to replicate the work of another programmer from scratch, just using the specifications. Once I had finished developing my code, and produced a final data set, I needed to compare our results. The comparisons showed multiple discrepancies in variable attributes, and values. I referred back to the programmer's original code to find the source of these discrepancies. This required reviewing and debugging the original code step by step to resolve the differences.

Debugging my own code was quite straightforward. Since I was the author of the code, I was able to read and understand it. In short, because I had written it, I was familiar with its nuances. Debugging the original code was more of a challenge. Reviewing the log could find syntax errors, and warnings. There weren't any syntax errors. There were warnings, but only minor ones. The source of the discrepancies went beyond these warnings, and required careful reading and reviewing of code.

LINE SPACING

Figure 1 has a sample of code from the original SAS program which was written for this project. It resembles one long, continuous block of code. Each step of the program is stacked directly on top of the next. With the code organized this way, it was a challenge to figure out. I could see where the first data set was created. I got lost trying to track what happened to it from that point on.

Figure 1. Sample Original Code

```
    /****** DSQ records FOR 2007-2008 *******/
DATA DIET_SUPP_2 ;/*(KEEP=RESP_ID MECNO DSQ012); */
SET ANAL1.VIEW_DIETSUPP;
WHERE 218<=MECNO<=247;
RUN;
PROC SORT DATA=DIET_SUPP_2; BY RESP_ID;
RUN;
```

```
DATA TEMP11;
SET DIET_SUPP_2;
WHERE RESP_ID=253795;
RUN;
********  DSQ012 STATUS FROM DSQ FILE ;
DATA DIETSUPP_ALL_3;
MERGE DIET_SUPP_1 (IN=A)
DIET_SUPP_2 (IN=B);
     BY RESP_ID;
IF A;
DSQ012_=DSQ012;RUN;
```

RECOMMENDATION – Insert a space between blocks of code.

This seems to be a basic enough concept to think it wouldn't be overlooked. You might be surprised. If you separate your PROC and DATA STEPS with a line space, then you can distinguish one step from the next. Then you can tell how that one step is connected to the step that follows, in terms of data sets. Further, you can see clearly the beginning and ending point of each step. So, it assists the reader in understanding the data flow in the program.

The sample code in Figure 2 has line spaces inserted between blocks of code. The spaces allow each step to stand on their own. Each step can be distinguished separately and individually. Now the reader can tell that the program is comprised of four steps altogether, including three DATA STEPS, and one PROC step. Three new SAS data sets are being created in the code.

Figure 2. Sample New Code

```
/****** DSQ records FOR 2007-2008 *******/
DATA DIET_SUPP_2 ; /*(KEEP=RESP_ID MECNO DSQ012); */ SET
ANAL1.VIEW_DIETSUPP;
WHERE 218<=MECNO<=247;
RUN;

PROC SORT DATA=DIET_SUPP_2;
BY RESP_ID;
RUN;

DATA TEMP11;
SET DIET_SUPP_2;
WHERE RESP_ID=253795;
RUN;

********  DSQ012 STATUS FROM DSQ FILE ;
DATA DIETSUPP_ALL_3;
MERGE DIET_SUPP_1 (IN=A) DIET_SUPP_2 (IN=B);
BY RESP_ID;
IF A;
DSQ012_=DSQ012;
RUN;
```

The first data step creates a temporary SAS data set, DIET_SUPP_2.

```
DATA DIET_SUPP_2;
```

The next step is the SORT Procedure which sorts the DIET_SUPP_2 data set.

```
PROC SORT DATA = DIET_SUPP_2;
```

The third step creates a new temporary data set, TEMP11.

```
DATA TEMP11;
```

The fourth step produces DIETSUPP_ALL_3a, the third data set.

```
DATA DIETSUPP_ALL_3;
```

INDENTATION

Is there anything else we can do to further clarify the code? In Figure 2, you'll notice that no statements or lines have been indented. They're aligned directly beneath each other.

RECOMMENDATION – Indent statements at least two spaces from the left margin.

Each statement within a step should be indented at least two spaces from the left margin. This is just another simple modification to further clarify SAS code. Indenting the statements within a step highlights those statements. Once you do this, the reader knows what's happening to the data within that step. If it's a DATA STEP, you can understand how the data's being manipulated, i.e. variables being assigned or re-coded.

For instance, in Figure 3, you can see that a permanent SAS data set is read, and its records are subset to create a new temporary work data set. Then, the new data set is sorted on a specific variable (RESP_ID). In the next step, the same variable is used to further filter your data. The resulting set of data is output to a new data file.

Figure 3. New Code

```
/****** DSQ records FOR 2007-2008 *******/
DATA DIET_SUPP_2 ;      /*(KEEP=RESP_ID MECNO DSQ012); */
  SET ANAL1.VIEW_DIETSUPP;
  WHERE 218<=MECNO<=247;
RUN;

PROC SORT DATA=DIET_SUPP_2;
   BY RESP_ID;
RUN;

DATA TEMP11;
   SET DIET_SUPP_2;
   WHERE RESP_ID=253795;
RUN;

********  DSQ012 STATUS FROM DSQ FILE ;
DATA DIETSUPP_ALL_3;
  MERGE DIET_SUPP_1 (IN=A)   DIET_SUPP_2 (IN=B);
    BY RESP_ID;
    DSQ012_=DSQ012;
  IF A;
RUN;
```

COMMENTS AND COMMENT STATEMENTS

Comments are a way to document your code so that others, programmers and non-programmers alike, can understand what your code is doing. In Figure 4, the comments are stacked on top of the code. Written this way, it's hard to tell them apart from the code. They are mixed in with it. Your team leader and co-workers may get confused reading this code.

Figure 4. Code Sample

```
%include "\\network-server\code_folder\macros\setup.sas";
ods html close;
options ps=50 ls=120;
* income ;
*I.  Establish parameters;
%let sample_file=sample0910.sample2009_2010;
%let start_mecno=248;
%let end_mecno=%eval(&start_mecno+29);
```

RECOMMENDATION - Set comments to be 12 Tab spaces from the left margin.

Comments should complement your code. They should add to it, not detract from it. I recommend setting your comments 12 tabs from the left margin, with one tab set at 4 spaces. In Figure 5 is a program that is formatted accordingly. Under this rule, comments are separated from the code, and they stand out from it. With the code justified on the left, and the comments justified on the right, the reader can refer back and forth, as they review your program. If they are confused about anything on the left, they can refer to the comments on the right. This way the comments act as a guide for deciphering your code.

Figure 5. Code Sample

```
                                                /****** DSQ records FOR 2007-2008 *******/
DATA DIET_SUPP_2 ;
  SET ANAL1.VIEW_DIETSUPP;
  WHERE 218<=MECNO<=247;
RUN;

PROC SORT DATA=DIET_SUPP_2;
  BY RESP_ID;
RUN;

DATA TEMP11;
  SET DIET_SUPP_2; WHERE
  RESP_ID=253795;
RUN;
                                                ********  DSQ012 STATUS FROM DSQ FILE ;
DATA DIETSUPP_ALL_3;
  MERGE DIET_SUPP_1 (IN=A)    DIET_SUPP_2 (IN=B);
    BY RESP_ID;
    DSQ012_=DSQ012;
  IF A;
RUN;
```

CASE\CAPITALIZATION

There are differing opinions in the SAS user community concerning the proper case to use in coding SAS programs. In Figure 6, is a sample SAS program with an inconsistent application of uppercase and lowercase. It's not applied uniformly across the DATA Steps and PROCS. The first two PROC keywords are in uppercase, then the next two in mixedcase.

Figure 6: Sample code - Case

```
      /*   get correct MECS and potential RESP's   */
PROC FREQ DATA=Year.RESPS%substr(&year,3,2);
  TABLE MECNO*SELECT_RESP*PERSON_TYPE / LIST MISSING;
  TABLE AGEYR*GENDER / LIST MISSING;  /* test */
  TITLE "NHANES_PERSON -- &year";
RUN;

PROC SORT DATA=Year.RESPS%substr(&year,3,2) NODUPKEY;
      BY RESP_ID;
RUN;

Proc Freq data=ANAL1.VIEW_EC;
   Table ec: FS121 WH030e MC080e / list missing;
   where &first_mecno <= MECNO <= &Last_mecno and AGEYR<= 15;
  Title 'Check EC skips';
run;

Proc freq data=ANAL1.view_hu;
  Table hu: / list missing;
  Where &first_mecno <= MECNO <= &Last_mecno and AGEYR > 15;
  Title 'Check HU skips'; run;
```

The first two PROCS (FREQ, and SORT) are in all caps, with the next one in mixed case, and the last one in lower case. Data sets, librefs, and variables are also written in one case in one instance, and then another case in a later instance.

I think it's important for the purposes of interpreting and understanding your code to have a consistent use of case for SAS statements\keywords, and data set names throughout your program. I also think it makes a difference to use a different case for data sets, than for statements and keywords. It further distinguishes between the two, and allows others to keep track of data sets in your code.

RECOMMENDATIONS

1. UPPERCASE FOR DATA SET NAMES.

Data sets and variables are the central feature of your SAS code. They're really what you're concerned with. The procedures, functions, and other SAS language elements are tools that you use to shape or massage your data. Thus, I advocate using uppercase for data sets (see below) and variables to highlight them as the focal point.

Figure 7: Sample code - Uppercase

```
Data TEMP2;
  Set TEMP1;
Run;
```

2. MIXED CASE FOR STATEMENTS\KEYWORDS

Likewise, using mixed case for these other SAS language elements puts them in the background in your program. I propose using mixed case for SAS PROCS, statements, keywords, and options. The code sample below is formatted according to this style. It demonstrates the visual representation that is achieved by using this particular coding style.

Figure 8: Sample code - Mixed case

```
Proc Freq Data=Year.RESPS09;
  Table MECNO*SELECT_RESP*PERSON_TYPE
        AGEYR*GENDER / List Missing;
  Title "NHANES_PERSON -- &year";
Run;

Proc Sort Data=Year.RESPS09 Nodupkey;
  By RESP_ID;
Run;
```

CONCLUSION

For this project, the program that I wrote from scratch incorporated these recommended elements of style. It was adopted as the default template program for future years of the project. It was precisely these stylistic aspects of the program that led to it being accepted. I strongly suggest using these stylistic techniques when coding your SAS program. You'll save valuable time for others that they can then devote to other high priority tasks.

ACKNOWLEGEMENTS

The author would like to thank Maribeth Johnson, SAS Global Forum 2016 Quick Tips Session Coordinator, Jennifer Waller, SGF 2016 Conference Chair, and the SGF 2016 Conference team and committee for accepting my abstract and paper, and organizing a great conference.

TRADEMARK CITATION

Jay Iyengar is an independent consultant, trainer and SAS Certified Professional. He is currently the co-organizer of the Chicago SAS Users Group, WCSUG. He has been using SAS since 1997. His industry experience includes Government, Healthcare and Clinical\Pharmaceutical. He presented a paper at MWSUG 2014, entitled 'Navigating the Data Universe Using the Starship Enterprise Guide'. He has also presented papers at the Northeast SAS Users Group (NESUG), and Southeast SAS Users Group (SESUG) conferences. He regularly attends meetings of the Wisconsin-Illinois SAS Users Group (WIILSU).

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Jay Iyengar
Data Systems Consultants LLC
Oak Brook, IL 60523
Email: IyenJ@aol.com
LinkedIn: http://www.linkedin.com/pub/jay-iyengar/31/897/577

APPENDIX

I. Original code re-formatted according to suggested elements of style.

```
                                          /****** DSQ records FOR 2007-2008 *******/
Data DIET_SUPP_2;
  Set ANAL1.VIEW_DIETSUPP;
  Where 218<=MECNO<=247;
Run;

Proc Sort Data=DIET_SUPP_2;
  By RESP_ID;
Run;

Data TEMP11;
  Set DIET_SUPP_2;
  Where RESP_ID=253795;
Run;

                                          ********  DSQ012 STATUS FROM DSQ FILE;
Data DIETSUPP_ALL_3;
  Merge DIET_SUPP_1(In=A) DIET_SUPP_2(In=B);
    By RESP_ID;
      DSQ012_=DSQ012;
  If A;
Run;
```