SAS Programming Basics

As with any programming language, the syntax for SAS is very specific. Small changes (for example, changing a semicolon to a period) can have dramatic consequences. A SAS program is comprised of SAS statements, which may be one or more lines in length. The beginning of a SAS statement usually contains a key word. These are indicated by bold letters in the program below. The end of a SAS statement is always marked by a semicolon.

In addition to the statements that instruct SAS to perform specific functions, a SAS program can contain comments. There are two ways to do this:

- start the line with an asterisk (*) and end with the a semicolon
- start the line with a slash and asterisk (/*) and end with asterisk and slash (*/)

If your comment fits on a single line, you can use either of these methods to make a comment. If your comment is two or more lines long, then you must use the second method. I encourage you to include comments in your SAS programs. This will help you understand what the code is doing, and it will help me if I need to examine your code.

The program editor in SAS Studio uses color-coding to indicate other key components of a SAS program. We will encounter

- yellow highlighting (orange on some monitors) indicates the data that the program will use
- light blue letters indicate key words for optional instructions
- green letters indicate comments
- bold letters indicate key words that begin a SAS statement

Here is a small SAS program to get us started:

```
/* A First SAS Program */

DATA example1;
INPUT Student$ Exam1 Exam2;
DATALINES;
JHK 80 83
CJD 55 63
BAO 57 .
RAL 62 71
WXY 75 87
AMT 91 92
RCG 83 91
MRR 59 63
CLE 93 99
MON 81 84
;

* Print the data to make sure it was read correctly;

PROC PRINT DATA = example1;
 VAR Student Exam1 Exam2;
 RUN;

* Generate summary information about the data;

PROC MEANS N MEAN STD STDERR LCLM UCLM;
 VAR Exam1 Exam2;
 RUN;
```

To help you understand SAS programs, SAS keywords are shown in upper case, and names of variables and datasets are shown in lower case. None of these are case-sensitive to SAS – everything could be lower case (or upper case) and the program would be exactly the same.

This program does three things:

1. Create a SAS data set that contains three variables (Student, Exam1 and Exam2).
2. Print the dataset.
3. Calculate (and print) various summary statistics for the two numeric variables (Exam1 and Exam2).

More information about this code is given on the following page.

```
DATA example1;
INPUT Student$ Exam1 Exam2;
DATALINES;
```

DATA creates a data set.

'example1' is an arbitrary optional name for the data set.

All SAS statements end in a semi-colon.

INPUT gives names to the variables. Variable names begin with a letter and may contain letters, numbers, and the underline, with no spaces between. Variable names are *not* case sensitive.

The $ after a variable name indicates that this variable contains non-numeric data. The $ is not part of the name.

DATALINES tell SAS that data come next.

```
JHK 80 83
CJD 55 63
BAO 57 .
RAL 62 71
WXY 75 87
AMT 91 92
RCG 83 91
MRR 59 63
CLE 93 99
MON 81 84
;
```

The data values are separated (i.e., delimited) by spaces. Make sure these are not tabs.

Character data cannot contain spaces. For instance, if the variable is "state", New York would have to be NY or NewYork or New_York or some other designation with no spaces.

The period (.) is the placeholder for missing values.

The yellow background reminds us that this is data.

```
PROC PRINT DATA = example1;
VAR Student Exam1 Exam2;
RUN;
```

PROC stands for procedure.
VAR tells SAS which variables to print.
RUN tells SAS to execute the statements.

```
PROC MEANS N MEAN STD STDERR
           LCLM UCLM;
VAR Exam1 Exam2;
RUN;
```

The MEANS procedure computes various summary statistics for numeric data.

We are asking SAS to calculate the sample size (N), the mean, the standard deviation (STD), the standard error of the mean (STDERR), and the lower and upper 95% confidence interval limits for the mean (LCLM and UCLM).

VAR tells SAS which numeric variables to include in the computations.

Here is the output that was generated

The output from the program is on the RESULTS tab in SAS Studio. If you want to save this output, you have several options:

1. Copy the contents of the RESULTS tab and paste it into a Word document.
2. When the contents of the RESULTS tab are visible, use the document icon to download the entire contents as an RTF file. The formatting will be "messed up", but the file can be opened and edited with any word processing program.
3. Download the output as an HTML file. This preserves the formatting, but you will need to use an HTML editor to make any changes.

Here are the results of proc print and proc means. (The data step does not produce any output.)

You should ALWAYS print the data the first time you read it into SAS.
Minor problems, like an extra space or a missing period, can create a nonsensical data set in SAS.

Obs	Student	Exam1	Exam2
1	JHK	80	83
2	CJD	55	63
3	BAO	57	.
4	RAL	62	71
5	WXY	75	87
6	AMT	91	92
7	RCG	83	91
8	MRR	59	63
9	CLE	93	99
10	MON	81	84

The MEANS Procedure

Variable	N	Mean	Std Dev	Std Error	Lower 95% CL for Mean	Upper 95% CL for Mean
Exam1	10	73.6000000	14.2766320	4.5146675	63.3871127	83.8128873
Exam2	9	81.4444444	12.9432522	4.3144174	71.4953801	91.3935088

A formatting note

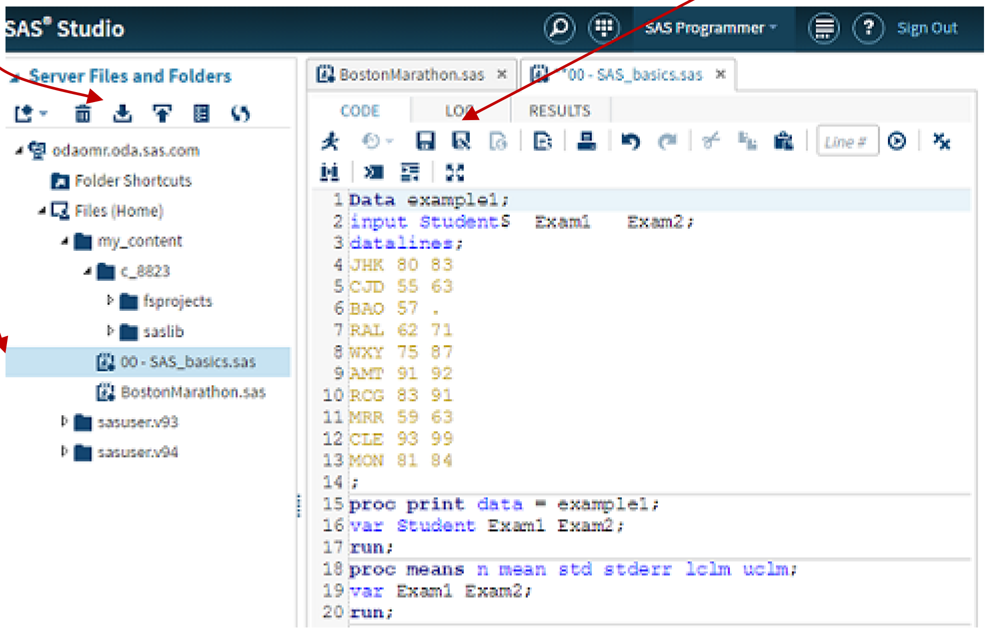
SAS does some very strange formatting with their tables. Most likely you will need to do some re-formatting to make the results legible. For example, I copied the results from SAS Studio and pasted it into my word processor. Then I had to change the line spacing to single-space and then center the two tables. You will need to make similar changes to any SAS output you want include in a word processing document.

File Management

SAS Studio operates through your web browser, and does not recognize any files you have stored on your local machine. If you use any of the SAS Studio icons to save a program (or output from a program), the file will be stored on the SAS server and not on your local machine. Similarly, if you use a SAS Studio icon to open a file, the file must be on the SAS server.

All of the SAS program files that I provide will have the ".sas" extension. As you encounter these files throughout the semester, you should download these files from Canvas to your local machine. To run these programs, you can drag and drop the file from your local machine to the SAS Studio program window. You can edit these files directly in the SAS Studio program window, but to save your changes you will need to do two things:

1. Save the program file to the SAS server, using either the 'save' or 'save as' icons:
2. Download this file to your local machine
    a. Click on the file you want to download
    b. Click the download icon



```
Data example1;
input Student$ Exam1 Exam2;
datalines;
JHK 80 83
CJD 55 63
BAO 57 .
RAL 62 71
WXY 75 87
AMT 91 92
RCG 83 91
MRR 59 63
CLE 93 99
MON 81 84
;
proc print data = example1;
var Student Exam1 Exam2;
run;
proc means n mean std stderr lclm uclm;
var Exam1 Exam2;
run;
```

You can also use any text editor (e.g., Notepad, Wordpad) to create and edit SAS programs on your local machine. When you save these files to your local machine, remember to use the ".sas" extension. To edit an existing SAS file without using SAS Studio, right-click on the file name and "open with" either Notepad or Wordpad. Do not use Word or any other word processing program to edit SAS program files. Invisible formatting symbols (e.g., tab characters) can really mess up a SAS program.

Debugging SAS Programs

It is extremely rare that any person can write a SAS program without making at least one mistake. This is true even for experienced SAS programmers. I will be providing SAS programs to accompany the examples in this course, but you will be required to modify this code for the assignments. I anticipate that there will be mistakes in your modifications, and you will need to find the mistakes and correct them. This process is called debugging.

There are two basic things you need to do for every SAS program.

1. Always print the data.
   The first time you create a SAS dataset (with a DATA statement), you should print the data. Examine the data table in the SAS output file to make sure SAS is using the correct data. Once you have determined that the data is being read correctly, you can delete the print statements.

2. Always look at the log file.
   To see this file, click on the LOG tab after you have run a SAS program. Most of this file contains system processing information (like memory and cpu time), which you can ignore. You need to look for error messages, warnings and notes.

Example 1.

Here is an example of what can go wrong:

SAS code for Example 1:

```
DATA wrong;
INPUT A B C;
DATALINES;
Mary 10 20
Jose 11 33
Bill 3 22
Fred 50 2
;
PROC PRINT DATA=wrong;
RUN;
```

Example 1 output:

Obs	A	B	C
1	.	10	20
2	.	11	33
3	.	3	22
4	.	50	2

Note that the numeric values for B and C are correct, but there are dots for all the values of A. This means SAS could not correctly interpret the values for A. (A dot in a SAS dataset indicates a missing value.) The problem here is that the variable A contains alphabetic information, so there needs to be a dollar sign after A in the INPUT statement.

The log file for this example is shown below. There are no error messages or warnings in this particular log file but there are several "notes" about invalid data. This is your indication that something is wrong with the data. The log file also tells you exactly where the invalid data is located (the first one is on line 57, columns 1-4). This information can help you pinpoint where the mistakes might be in your code.

Log file for Example 1:

```
1          OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
53
54         DATA wrong;
55         INPUT A B C;
56         DATALINES;

NOTE: Invalid data for A in line 57 1-4.
RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6
57         Mary 10 20
A=. B=10 C=20 _ERROR_=1 _N_=1
NOTE: Invalid data for A in line 58 1-4.
58         Jose 11 33
A=. B=11 C=33 _ERROR_=1 _N_=2
NOTE: Invalid data for A in line 59 1-4.
59         Bill 3 22
A=. B=3 C=22 _ERROR_=1 _N_=3
NOTE: Invalid data for A in line 60 1-4.
60         Fred 50 2
A=. B=50 C=2 _ERROR_=1 _N_=4
NOTE: The data set WORK.WRONG has 4 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      user cpu time       0.00 seconds
      system cpu time     0.00 seconds
      memory              763.93k
      OS Memory           24224.00k
      Timestamp           06/22/2016 01:50:15 AM
      Step Count                        16  Switch Count  66
      Page Faults                       0
      Page Reclaims                     346
      Page Swaps                        0
      Voluntary Context Switches        213
      Involuntary Context Switches      0
      Block Input Operations            0
      Block Output Operations           264

61         ;
62         PROC PRINT DATA=wrong;
63
64         OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
75
```

Example 2.

To correct the code for Example 1, put a dollar sign ($) after A in the input statement. Here is the corrected code and output:

SAS code for Example 2:

```
DATA right;
INPUT A $ B C;
DATALINES;
Mary 10 20
Jose 11 33
Bill 3 22
Fred 50 2
;
PROC PRINT DATA=right;
run;
```

Example 2 output:

```
Obs     A     B    C
  1  Mary    10   20
  2  Jose    11   33
  3  Bill     3   22
  4  Fred    50    2
```

Example 3.

Here is another example that illustrates a common mistake.

SAS Code for Example 3:

```
DATA example3;
INPUT A $ B C;
DATALINES;
Mary   10   20
Jose   11   33
Bill   3    22
Fred   50   2
;
PROC PRINT DATA=example3;
run;
```

Example 3 output:

```
Obs  A        B  C
  1  Mary 10  .  .
```

This particular mistake can be quite mystifying at first. There is apparently nothing wrong with the code, but the first value for A should be “Mary”, not “Mary 10”. In addition, the values for B and C are missing and there are no data values for Jose, Bill or Fred. The log file, shown on the next page, warn us about invalid data , but it is complaining about the values for B and C and does not indicate any difficulties with Jose, Bill or Fred.

The error in this program is that the data values are separated by tabs, not spaces. SAS expects the values to be separated by spaces. Even though we cannot see the tabs, SAS can “see” them, and it is trying to interpret the tab characters as part of the data. Since “Mary” and “10” are not separated by a space, SAS interprets both of these as the first value for A, then it gets confused about the values for B and C.

Log file for Example 3:

```
 1          OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
 53
 54         DATA example3;
 55         INPUT A $ B C;
 56         DATALINES;

 NOTE: Invalid data for B in line 58 1-10.
 NOTE: Invalid data for C in line 59 1-4.
 RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--

 59  CHAR   Bill .3.22
     ZONE
46662030332222222222222222222222222222222222222222222222222222222222222222222222
     NUMR
29CC0939220000000000000000000000000000000000000000000000000000000000000000000000
 NOTE: Invalid data errors for file CARDS occurred outside the printed range.
 NOTE: Increase available buffer lines with the INFILE n= option.
 A=Mary10 B=. C=. _ERROR_=1 _N_=1
 NOTE: LOST CARD.
 61         ;
 A=Fred50 B=2 C=. _ERROR_=1 _N_=2
 NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
 NOTE: The data set WORK.EXAMPLE3 has 1 observations and 3 variables.
 NOTE: DATA statement used (Total process time):
       real time           0.00 seconds
       user cpu time       0.00 seconds
       system cpu time     0.00 seconds
       memory              782.21k
       OS Memory           24224.00k
       Timestamp           06/22/2016 02:08:18 AM
       Step Count                        40  Switch Count  68
       Page Faults                       0
       Page Reclaims                     383
       Page Swaps                        0
       Voluntary Context Switches        221
       Involuntary Context Switches      0
       Block Input Operations            0
       Block Output Operations           264

 61         ;

 62         PROC PRINT DATA=example3;
 63
 64         OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
 75
```

<u>Example 4.</u>

To correct the mistake in Example 3, you have two options. First, you could edit the program and change all the tabs to spaces. This is probably the best option if the dataset is small. But this could be time-consuming if the dataset is large. A second way to solve this problem is to tell SAS that the data values are separated by tabs. This is done with an INFILE statement and the DLM option (which is an abbreviation for delimiter). The additional line of code is highlighted in yellow below.

```
DATA example4;
INFILE DATALINES dlm='09'x;
INPUT A $ B C;
DATALINES;
Mary    10    20
Jose    11    33
Bill    3     22
Fred    50    2
;
PROC PRINT DATA=example4;
run;
```

Obs	A	B	C
1	Mary	10	20
2	Jose	11	33
3	Bill	3	22
4	Fred	50	2

<u>Example 5.</u>

To save space, sometimes multiple rows of a dataset are combined on one line. SAS will only read the first one unless you tell it to read more. This is done by putting '@@' (without the quotes) at the end of the input statement. In the code below, Example 5a does NOT have '@@', and George is the only one in the dataset. Example 5b DOES include '@@', so all four are in the dataset. For both examples, the data values are separated by spaces.

```
DATA example5a;
INPUT Beatle $ number;
DATALINES;
George 35  Paul 12  Ringo 23  John  66
;
PROC PRINT DATA=example5a;
RUN;
```

Obs	Beatle	number
1	George	35

```
DATA example5b;
INPUT Beatle $ number @@;
DATALINES;
George 35  Paul 12  Ringo 23  John  66
;
PROC PRINT DATA=example5b;
RUN;
```

Obs	Beatle	number
1	George	35
2	Paul	12
3	Ringo	23
4	John	66

Example 6.

This example illustrates what is probably the most common mistake for new SAS programmers. This code does not generate any output. Can you find the mistake?

```
DATA example6;
INPUT Beatle $ number @@
DATALINES;
George 35  Paul 12  Ringo 23  John  66
;
PROC PRINT DATA=example6;
RUN;
```

If you cannot see the mistake, perhaps the log file will help. The log file, shown on the next page, gives an error message for the line that starts with George, but the biggest clue comes from the second error message: `ERROR: No DATALINES or INFILE statement.`

But we clearly have a DATALINES statement in our program, so why doesn't SAS see it???? Look at the line in our program immediately preceding the DATALINES statement. This is the INPUT statement. Do you see what is wrong with the INPUT statement? Answer: It is missing a semicolon.

Every SAS statement must end with a semicolon.

Forgetting a semicolon is like driving a car with no brakes – you never know where you will end up. Omitting a single semicolon is pretty much guaranteed to send SAS into la-la land and generate multiple error messages.

As a final note, I want to emphasize that I do not expect you to become an expert SAS programmer. SAS is a vast and complicated software system, and we will use only a tiny part of it. Also, I will provide sample code for everything we do, and I encourage you to use this code and adapt it for specific assignments. For this course, SAS is merely a computational tool. I do not want it to become an obstacle. If you ever have any questions about SAS (or anything else in this course), please email me. I usually respond within a few hours.

Log file for Example 6:

```
 1          OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
 53
 54         DATA example6;
 55         INPUT Beatle $ number @@
 56         DATALINES;
 57         George 35  Paul 12  Ringo 23  John  66
            ______
            180
 ERROR 180-322: Statement is not valid or it is used out of proper order.

 58         ;

 ERROR: No DATALINES or INFILE statement.
 NOTE: The SAS System stopped processing this step because of errors.
 WARNING: The data set WORK.EXAMPLE6 may be incomplete.  When this step was
stopped there were 0 observations and 3 variables.
 NOTE: DATA statement used (Total process time):
       real time           0.00 seconds
       user cpu time       0.00 seconds
       system cpu time     0.00 seconds
       memory              626.50k
       OS Memory           24480.00k
       Timestamp           06/22/2016 03:51:05 AM
       Step Count                        82  Switch Count  52
       Page Faults                       0
       Page Reclaims                     323
       Page Swaps                        0
       Voluntary Context Switches        154
       Involuntary Context Switches      0
       Block Input Operations            0
       Block Output Operations           264

 59         PROC PRINT DATA=example6;
 60         RUN;

 NOTE: No observations in data set WORK.EXAMPLE6.
 NOTE: PROCEDURE PRINT used (Total process time):
       real time           0.00 seconds
       user cpu time       0.00 seconds
       system cpu time     0.01 seconds
       memory              448.18k
       OS Memory           24480.00k
       Timestamp           06/22/2016 03:51:05 AM
       Step Count                        83  Switch Count  22
       Page Faults                       0
       Page Reclaims                     51
       Page Swaps                        0
       Voluntary Context Switches        32
       Involuntary Context Switches      0
       Block Input Operations            0
       Block Output Operations           0
```