

Introduction to SAS Programming¹

Introduction

What is SAS?

The SAS System (a.k.a. "SAS") is a popular set of industrial and educational use software tools, which allow you to access, manage, present, and analyze data. It runs on many different computer platforms and is designed to work similarly on different operating systems. Note that we will cover only Windows SAS in this course. SAS is organized into a number of modules, called products. These have names like base SAS, SAS/STAT (statistics), SAS/IML (matrices), and SAS/INSIGHT (interactive data analysis). Your SAS program will call the appropriate product for you, however; the only times you will need to know what feature is associated with what product is when you are looking at the documentation, and if you are doing a partial install of SAS to save space and need to know which products you may omit. While these products are licensed separately, the UNC site license covers most of them. The SAS System is produced by the SAS Institute in Cary, North Carolina, and is used in over 120 countries, at over 31,000 sites, and by an estimated 3.5 million users.

Versions of SAS

The versions of SAS currently in mainstream use are versions 8, 7, and 6.12, although version 6.04 is also still around in limited amounts. The main jump in SAS design came between versions 6.12 and 7, with version 8 adding only a few new features to version 7. This document was written using SAS 8, although most of these ideas will hold for all versions of SAS. Note that we will be using version 8 in this course.

How to get SAS at UNC

SAS is available at UNC in two forms:

- SAS off the UNC [statistical server StatApps](http://help.unc.edu/statistical/statapps.html)²
- SAS for Windows

For information on using SAS on StatApps, see the ATN [SAS documentation](http://help.unc.edu/statistical/statapps.html)³.

SAS for Windows can be obtained through [Software Acquisitions](http://help.unc.edu/software/)⁴.

¹ <http://help.unc.edu/statistical/applications/sas/introsasprog.html>

² <http://help.unc.edu/statistical/statapps.html>

³ <http://help.unc.edu/statistical/applications/sas>

⁴ <http://help.unc.edu/software/>

SAS Online Documentation

Starting with version 7, SAS offers its entire documentation in web format. This can be accessed by campus users at <http://statweb.unc.edu/onldoc.htm>.

Throughout this document will be links to specific pages of this documentation; however, in order to begin at the main screen where searches can be performed, you must access the online documentation using the previous link. In Windows, the online documentation can also be accessed from Help/Books and Training/SAS OnlineDoc on the Windows command bar.

SAS for Windows

Tour of the SAS Windows

SAS has six [windows](#)⁵ that you will use:

- Enhanced Editor: used to input, edit, and submit SAS programs. SAS key words are color coded and statements in SAS steps are collapsible. This window is new in version 8.
- Program Editor: also used to input, edit, and submit SAS programs, except without all the formatting in the Enhanced Editor. This window does not open automatically in version 8, but was the default editor window in versions 7 and below.
- Log: contains notes about a SAS program you submit including any errors, along with a copy of the actual program itself.
- Output: displays any printable results your SAS program generates.
- Results: displays an outline of all output produced in a SAS session (a single SAS invocation). Using this window, you can print or save certain output, but not others. Note that versions 6.12 and below do not have this window.
- Explorer: allows you to easily manage and view all your SAS files and libraries (libraries are covered later in this document). Note that versions 6.12 and below do not have this window.

Comments About These Windows

- Choices in the command bar at the top of the screen may change depending on which window you have highlighted. If you do not see the choice you want, check to see that you are in the appropriate window.
- You can minimize, maximize, resize, or close these windows like any other window in a Microsoft Windows environment. A special note about the results and explorer windows: these windows are **docked** when you

⁵ <http://statweb.unc.edu/win/index.htm>

invoke SAS, which means you can't minimize them. To undock them, select `Window/Docked` in the command bar.

- To restore a window to the screen after closing it, go to `View` in the command bar and select the appropriate window.
- You can right click with your mouse to get the same menus as in the command bar.
- You can customize your environment according to your own personal preferences. Look at `Tools/Customize` in the command bar to do this.
- You submit a SAS program by clicking on the running man icon in the command bar. If you are using the program editor window, your code will disappear when you submit it. To get it back, go to `Run/Recall Last Submit` in the command bar in versions 8 and 7 and `Locals/Recall Text` in the command bar in version 6.12.
- SAS has built-in help pages for each of these windows. These are definitely worth looking through if you are completely new to this software. These can be found by selecting `Help/Using This Window` in the command bar.

Writing SAS Programs

SAS Language Conventions

- A SAS program is a series of **steps** where each **step** is composed of a group of SAS statements executed in a particular order to do something. SAS programs execute statement by statement in the order in which the statements are typed.
- All SAS statements must end with a semi-colon. Omitting semi-colons is probably the most common and easy-to-fix SAS programming error-- watch out for this!
- Blanks or other special characters separate [words](#)⁶ in SAS statements.
- SAS statements can be in uppercase or lowercase or can alternate between the two. Use what looks best to you.
- SAS statements can continue onto the next line, so long as you don't split a word between lines.
- SAS statements can be on the same line as other SAS statements.
- SAS statements can start anywhere on the screen. Again, use what looks best to you.

As an example, the following programs are equivalent:

```
statement one;          STATEMENT one;      statement one; statement two;
statement two;          StAtEMEnt tWo;      statement
statement three;       statement THREEe;    three;
```

⁶ <http://statweb.unc.edu/lgref/z1031075.htm>

The Two Types of SAS Steps

Although there are a few exceptions, the only two types of SAS steps you will encounter are **data** steps and **proc** steps. Combinations of these will form all your SAS programs.

A **data** step creates or modifies a SAS dataset. Its first statement is of the form `data dataset_name`. Data steps have a built-in loop; i.e., SAS executes the statements in the data step for the first observation, then for the second, etc., until the last observation. Understanding that this occurs is crucial in writing statements in data steps.

A **proc** step usually analyses or processes data in some way, e.g. to make a graph or report. Its first statement starts with `proc` followed by the name of the procedure you're using, e.g., `print`. These procedures are prewritten by the folks at the SAS Institute and do a variety of useful things.

The last statement in each of these types of steps should always be `run;`. SAS reads statements until it finds a `run` statement or another data or proc step and then executes the preceding statements. Thus, you don't have to use `run` statements to execute SAS steps, but it is generally a good idea, since you won't always have another data or proc step following.

Examples of each of these:

data step

```
data mydata;  
statement one;  
statement two;  
etc;  
...  
run;
```

proc step

```
proc procedure_name;  
statement one;  
statement two;  
etc;  
...  
run;
```

How to Put Comments Into Your Program

Generally, it is considered good programming to include [comments](#)⁷ (i.e., statements that are not executed) throughout your SAS program so that any person could look at it and understand what you were trying to accomplish (or in most cases, so **you** can look back at old work and understand what you were trying to accomplish!). Comments can appear as stand-alone statements or as statements within data or proc steps. The following are the two ways to write comments:

```
Put your comment here;  
/*Put your comment here*/
```

⁷ <http://statweb.unc.edu/lgref/z0289375.htm>

As you can see, the first begins with an asterisk and ends with a semi-colon, and the second begins with a slash asterisk and ends with an asterisk slash. Which you use is a matter of preference, but note that the second can include comments with semi-colons (e.g., statements you don't want to execute but don't want to delete).

Example:

```
/*The following is a data step*/  
data mydata;  
statement one;  
statement two;  
etc;  
...  
run;
```

SAS Datasets

In order to use your data in SAS, it must be in the form of a SAS dataset. We'll learn later some ways of getting your data into the form of a SAS dataset, but for now, assume it is.

In general, a SAS dataset consists of a descriptor portion and a data portion. The descriptor portion contains information about the dataset such as the number of variables, their names and types, and the number of observations. The data portion contains the actual data arranged with variables in columns and observations in rows. SAS can handle up to 32,767 variables and as many observations as the memory of your computer will allow.

	Variable			
Observation				

SAS variables can be of two types, **numeric** or **character**. Numeric variables contain data that are numbers. They can be either positive or negative and with any number of decimal places and can also contain E for scientific notation. You can perform all the regular mathematical operations on these. Character variables contain data that are character strings. Note that character variables could contain numbers; the difference is just in how SAS treats them. Unlike numeric variables, you cannot perform mathematical functions on character variables (even if they contain numbers). An example of a common character variable whose data contains numbers is some sort of ID variable. In general, it doesn't make much sense to perform mathematical operations on values of an ID variable. Certain

types of statements apply only to numeric variables, and others only to character variables, so you must be aware of the variable types you are working with.

Missing Data

Sometimes some of your data will be missing. SAS represents numeric missing data with a period and character missing data with a blank. Looking at how missing data are represented for a particular variable is an easy way of identifying if a variable is numeric or character without having to view the descriptor information for the dataset.

For example, suppose we have the following dataset:

ID	Name	Age	Zip Code
01227	Anne	23	78250
47314	Steve	53	
74103	Mike	.	90210

By looking at values for missing data, we can see that `Age` is a numeric variable and `Zip Code` is a character variable.

SAS Naming Conventions (for variables and datasets)

- [Names](#)⁸ can be at most 32 characters long. In versions 6.12 and lower, names could be at most 8 characters long.
- Names must start with a letter or underscore.
- Names can contain only letters, numbers, or underscores.
- SAS does not distinguish between upper- and lowercase letters in naming; e.g., the variable name `temp` refers to the same variable as the names `TEMP` or `Temp`. A note about this: when printing a dataset, SAS prints the name of a variable using the case it had when it was first introduced. You can always change this in the print procedure with appropriate statements, but you should just be aware of this.

Examples of illegal names:

<code>lstrun</code>	Does not begin with a letter or underscore
<code>bodyfat%</code>	Contains an illegal character
<code>city_of_residence _in_May_1990</code>	Too long

⁸ <http://statweb.unc.edu/lgref/z1031056.htm>

More Variable Attributes

- Along with type (numeric or character), each variable has a **length** associated with it, where length is the number of bytes used to store each value of the variable.
- Values of [numeric variables](#)⁹ are stored in 3-8 bytes with the default being 8 bytes. Shorter lengths should be used only for variables with entirely integer values. Nonintegers stored in less than 8 bytes will be truncated and will thus lose precision.

Length (Bytes)	Largest Integer Represented Exactly
3	8,192
4	2,097,152
5	536,870,912
6	137,438,953,472
7	35,184,372,088,832
8	9,007,199,254,740,992

- The default length of character variables is also 8 bytes and corresponds directly to the number of characters in a the value of the variable. Values of a character variable can be up to 32,767 characters long and can thus have lengths up to 32K. It is a good idea to look at your dataset and find out the length of the longest character string in all values of a particular character variable and then set the length of the variable to this. For large datasets, this may result in a considerable reduction of storage space needed for the dataset.
- Variables have **formats** and **informats** associated with them. [Formats](#)¹⁰ control how values of a variable are displayed when the dataset is printed (e.g., you might want to display 52.5 as \$52.50). [Informats](#)¹¹ control what format the data is in when you enter it; i.e. how SAS reads in your data. These particularly have to be used if you are entering data with special characters, such as "\$". We'll see examples of how to use these shortly.

Dates in SAS

SAS stores dates as the number of days since January 1, 1960 (picked somewhat arbitrarily). You apply a format to them to make them recognizable, 08/15/1999 or 15AUG1999 for example. This means that dates are basically just numbers until you apply a format to them. Thus, SAS considers date variables numeric, which means you could perform all the same mathematical functions on them as you would on other numeric variables, although in most cases, these operations

⁹ <http://statweb.unc.edu/win/numvar.htm>

¹⁰ <http://statweb.unc.edu/lgreg/z0309859.htm>

¹¹ <http://statweb.unc.edu/lgreg/z0309877.htm>

don't make any sense. Informat must be used when reading in dates (since, of course, when entering dates into a dataset, you don't want to have to enter the number of days since January 1, 1960!). A note: of course, when reading in dates, you could store them as a character strings, but then you wouldn't be able to use any of the collection of date functions in SAS (we'll see these later).

All SAS dates between between 1700 A.D. and 2200 A.D. contain five or less digits, hence values of all date variables can be stored in four bytes.

Permanent and Temporary Datasets

In SAS, files are stored in **libraries**. A library is basically just a pointer to some directory on the computer SAS is running on. Names of libraries are called librefs. There are two common ways to specify a library:

- Submit the statement `libref name_of_lib "directory";`. For example:

```
libref cdrive "c:\";
```

- In Windows SAS, click on the libraries icon in the explorer window. Then, right click with your mouse and select `New`. You can then fill in the necessary information. Clicking "Enable at Startup" will allow this library to be available to you in future invocations of SAS.

Library names can be at most 8 characters long, but otherwise follow the same rules as variable and dataset names.

In SAS, permanent datasets, those which are still available to you after you terminate your SAS session, are referenced with names of the form `libref.dataset_name`, where `libref` is the name of the library where the dataset is located, as described above. For example, if the dataset `introsas` was located in `c:\sasdocs`, you would give `c:\sasdocs` a name, say `mydocs`, using one of the methods above, and then the dataset would be referenced as `mydocs.introsas`.

SAS also has a special library, the `work` library, which points to a `TEMP` directory somewhere in your SAS installation files. To reference a dataset here, using the above logic, you would use the name `work.dataset_name`. But actually, you can do it an even easier way. You can just use the name `dataset_name`; if a dataset is referenced without a `libref`, `work` is the understood library. What's special about the `work` library is that datasets here are temporary--they are deleted when you exit your SAS session.

Actual Dataset Extensions

SAS uses different extensions when saving permanent datasets in the different versions of SAS. Note that when calling datasets in SAS, you do **not** give an extension. Some of the main extensions are:

Version	Extension
8.0	.sas7bdat or .sd7
7.0	.sas7bdat or .sd7
6.12 for Windows	.sd2
6.12 for Unix	.ssd01
6.04 for PC/DOS	.ssd

How to Get Your Data Into SAS

The following is a list of the main ways of getting data into SAS. Discussion of these methods is beyond the scope of this document; more information about each can be found in the [SAS Online Documentation](#)¹².

Converting Datasets From Other Software Packages Into SAS Datasets

- Use DBMS/COPY, a dataset conversion package from Conceptual Software, Inc. See the [ATN DBMS/COPY documentation](#)¹³ for more information.
- Use [Dynamic Data Exchange \(DDE\)](#)¹⁴ or the [IMPORT procedure](#)¹⁵ in SAS.
- While in the other software application, convert your dataset to raw data (this option is almost always available). This can then be read into SAS as described below.

Reading Datasets From Other Software Packages Directly

- Use the [SAS/ACCESS](#)¹⁶ product.
- Use different data engines that are part of base SAS. Note: in specifying a library, you can also specify an engine (previously we used the default engine, the SAS version 8 engine). Engines for many popular database systems are available.

Entering Data Directly Into SAS Datasets

¹² <http://statweb.unc.edu/onldoc.htm>

¹³ <http://help.unc.edu/statistical/applications/dbmscopy/index.html>

¹⁴ <http://statweb.unc.edu/win/dde.htm>

¹⁵ <http://statweb.unc.edu/proc/z0332605.htm>

¹⁶ <http://statweb.unc.edu/accpc/index.htm>

- Use the Viewtable window included in base SAS, which allows you to enter your data in table form.
- Use the [SAS/FSP](#)¹⁷ product, which allows you to create a customized data entry screen.

Creating SAS Datasets From Raw Data Files

- Use the [IMPORT procedure](#)¹⁸ in SAS (available only in Windows SAS).
- Use certain statements in a data step to read in raw data (data contained in a text file). This method will be covered in depth in the next section.

Raw Data

Raw data can be **internal** or **external**. Internal raw data are data that are typed directly into your SAS program; external raw data are read in from a file. In the examples in this section, we will use the following data introduced earlier:

ID	Name	Age	Zip Code
01227	Anne	23	78250
47314	Steve	53	
74103	Mike	.	90210

Internal Raw Data

The following programs all create a temporary SAS dataset named `temp` by typing the above data directly into a SAS program:

Program One

```
data temp;
input id $ name $ age zip_code $;
datalines;
01227 Anne 23 78250
47314 Steve 53 .
74103 Mike . 90210
;
```

Program Two

```
data temp;
infile datalines dlm=' ';
input id $ name $ age zip_code $;
datalines;
01227,Anne,23,78250
47314,Steve,53,
74103,Mike, ,90210
```

¹⁷ <http://statweb.unc.edu/fsproc/index.htm>

¹⁸ <http://statweb.unc.edu/proc/z0332605.htm>

;

Program Three

```
data temp;
input id $ 1-5 name $ 7-11 age 13-14 zip_code $ 16-20;
datalines;
01227 Anne 23 78250
47314 Steve 53 .
74103 Mike . 90210
;
```

The only difference between these programs is how the data values in the program are separated. The first program separates values using **spaces**, the second program separates values using **commas**, and the third uses **column input**, a method where the program tells SAS which columns contain the values for each variable. Let's discuss each of these statements individually:

```
data temp;
```

```
infile options;
```

```
input list_of_variables;
```

```
datalines;
```

```
data values
```

Since we are creating a dataset, we use a data step. Thus, the first statement must be of the form `data dataset_name`. Note that we could have made this a permanent dataset. As we will see shortly, this statement is generally used to tell SAS the location of a file with external raw data, but here we can use the `datalines filename and dlm=','` option to tell SAS that the following internal raw data are separated by commas.

The `input` statement tells SAS the order of your variables. Note that character variables must be followed by a `$`. In program three, we also include the columns where values for each variable are found.

This statement tells SAS that the following statements will contain lines of raw data.

Separate data values in whatever way you like most. Note that missing data are all entered with a `.` regardless of what type the variable is (SAS will assign the appropriate missing value character for you). There are other ways you can enter missing data, but this works.

;

This statement is called a null statement and signifies the end of data input. The statement `run;` would also work.

External Raw Data

The statements for reading external raw data are much the same as for internal raw data. Now assume the example data is located in the external file `h:\temp\example.txt`. The following programs create a temporary SAS dataset `temp2` by reading in this file. Again, the differences between these programs reflect whether data values in the external file are separated by spaces or commas, or are arranged in columns for use in the column input style.

	Appearance of Input Dataset
Program One	
<code>data temp2;</code>	01227 Anne 23 78250
<code>infile 'h:\temp\example.txt';</code>	47314 Steve 53 .
<code>input id \$ name \$ age zip_code \$;</code>	74103 Mike . 90210
<code>run;</code>	
Program Two	
<code>data temp2;</code>	01227,Anne,23,78350,
<code>infile 'h:\temp\example.txt' dlm=',';</code>	47314,Steve,53,.,
<code>input id \$ name \$ age zip_code \$;</code>	74103,Mike,.,90210
<code>run;</code>	
Program Three	
<code>data temp2;</code>	01227 Anne 23 78250
<code>infile 'h:\temp\example.txt';</code>	47314 Steve 53 .
<code>input id \$ 1-5 name \$ 7-11</code>	74103 Mike . 90210
<code>age 13-14 zip_code \$ 16-20;</code>	
<code>run;</code>	

The only new statement we encounter in these programs is the `infile` statement, which as we mentioned before, tells where the file to be read is located.

Using Formats and Informats

Suppose you had the following data stored in the file `h:/temp/sales.dat`:

```
06/22/1995 Martha $2,031.90
12/25/1996 Natalie $4,591.01
09/28/1998 Victoria $1,993.24
04/12/1999 Sandy $2,540.67
05/08/1999 Jacqueline $2,357.42
```

We apply informats so this data is read properly and formats so the data is printed as we want:

```
data sales;
informat date mmdyy8. clerk $10. profit dollar7.2;
format date date7. profit dollar5.;
infile 'h:\temp\sales.dat';
input date clerk profit;
run;
```

Notes:

- The `format` and `informat` statements are of the form

```
(in)format variable1 (in)format1 variable2 (in)format2 etc.;
```

- You need not include all the variables in your dataset in these statements, only the ones you wish to impose an informat or format on.
- By specifying this `format` statement in a data step, these formats will be used every time you output your data. A `format` statement can also be placed in a proc step to specify formats lasting only for the duration of that procedure.
- A `$` was not needed after `clerk` in the `input` statement since the `informat` statement already specified `clerk` as a character variable. If `clerk` was not in the `informat` statement, this `$` would be necessary.
- The general form of formats and informats is `(in)formatw.d` for numeric variables and `$(in)formatw.` for character variables, where `w.` is the total width of the values of the variable and `d` is the number of decimal places. Note that although dates are numeric variables, clearly their formats and informats will not have decimal places. The following are some popular [formats](#)¹⁹ and [informats](#)²⁰:

Name	Example	Width Range	Default Width
<code>\$w.</code>	Displays character values with a width of <code>w.</code> Trims leading blanks.	1-32767	none
<code>\$CHARw.</code>	Displays character values with a width of <code>w.</code> Does not trim leading blanks.	1-32767	8 or length of variable
<code>DATEw.</code>	22JUN95	7-32	7
<code>MMDDYYw.</code>	06/22/1995	6-32	6
<code>DDMMYYw.</code>	22/06/1995	6-32	6
<code>DOLLARw.d</code>	\$2,032	2-32	6
<code>COMMAw.d</code>	2,031.90	1-32	1

¹⁹ <http://statweb.unc.edu/lgreg/z1263753.htm>

²⁰ <http://statweb.unc.edu/lgreg/z1239776.htm>

Some Simple Data Step Statements

This section contains a sample SAS data step with many basic [data step statements](#)²¹. This example should give you a good base from which to write your own data step statements. For this example, assume we have the following SAS dataset located at h:\temp\class.sas7bdat.

Name	Sex	DOB	HW1	HW2	HW3
John	M	11/19/1981	100	75	81
Brian	M	10/24/1980	89	93	97
Shannon	F	01/30/1981	99	94	50
Sherry	F	01/19/1976	82	100	100
Brandon	M	05/17/1980	77	83	59
Jennifer	F	09/13/1981	99	98	100
Jamie	M	03/01/1977	88	76	100
Michael	M	02/28/1979	91	88	93
Thomas	M	11/21/1979	54	0	25
Mary	F	10/07/1978	89	95	95
Kelly	F	06/27/1970	97	97	97
Gabriel	M	03/23/1977	95	90	91

Here's the example program:

```
libname examples "h:\temp";
libname hdrive "h:\";

data hdrive.class;
length age 3;
set examples.class end=eof;
where sex="F";
hw_avg=mean(hw1,hw2,hw3);
age=int((today()-dob)/365.25);
if hw_avg ge 90 then hw_grade='A';
else if hw_avg ge 80 then hw_grade='B';
else if hw_avg ge 70 then hw_grade='C';
else if hw_avg ge 60 then hw_grade='D';
else hw_grade='F';
retain j 0 hwsum1 0 hwsum2 0 hwsum3 0 overall_sum 0;
j=j+1;
array hw[7] hw1-hw3 hwsum1-hwsum3 overall_sum;
do i = 1 to 3;
    hw[i+3]=hw[i]+hw[i+3];
    hw[7]=hw[7]+hw[i];
end;
```

²¹ <http://statweb.unc.edu/lgreg/z1225397.htm>

```

if eof then do;
  fem_hw1_avg=hwsum1/j;
  fem_hw2_avg=hwsum2/j;
  fem_hw3_avg=hwsum3/j;
  fem_hw_avg=overall_sum/3/j;
end;
drop sex dob hw1 hw2 hw3 i j hwsum1 hwsum2 hwsum3
  overall_sum;
format hw_avg fem_hw1_avg fem_hw2_avg
  fem_hw3_avg fem_hw_avg 4.1;
label hw_grade="Homework Grade";
run;

```

Let's discuss each of these statements individually:

```

libname examples "h:\temp";
libname hdrive "h:\";

```

These statements assign the libraries `examples` and `hdrive`.

```

data hdrive.class;

```

This data step creates the permanent dataset `h:\class.sas7bdat`.

```

length age 3;

```

This statement sets the length of the variable `age` (soon to be created) to three bytes. In general, a `length` statement is of the form

```

length var1 length1 var2 length2 etc.;

```

A `length` statement must always occur before any of the variables in it are referenced. Thus, it is generally a good idea to always make it the second statement in a data step.

```

set examples.class end=eof;
where sex="F";

```

A `set` statement allows you to access data from another existing SAS dataset. Here we are creating the dataset `hdrive.class` from the existing dataset `examples.class`. The `end=eof` option creates an internal variable `eof` which equals 1 on the last observation and 0 otherwise. We will use this variable later to perform operations only on the last observation.

A `where` statement subsets data from the original dataset `examples.class`. In this case, the new dataset `hdrive.class` will contain only observations from `examples.class` where the variable `sex` equals `F`. A note: SAS **does** consider case when

referring to character strings; i.e., the string F is **not** the same as f. If you had written the where statement as where sex="f", none of the observations would have been included. Watch out for this.

```
hw_avg=mean(hw1, hw2, hw3);  
age=int((today()-date)/365.25);
```

These statements illustrate three built-in [functions](#)²² in SAS, the mean, int, and today functions, and how to create new variables in a dataset. In general, functions in SAS have the form

```
function_name(arguments)
```

Here the mean function calculates the mean of the variables hw1, hw2, and hw3 for each observation. We assign this value to the variable hw_avg by using the statement hw_avg=mean(...). The today() function (which has no arguments), returns today's date. We can calculate each subject's age (assigned to the variable age) by subtracting their birthday from today's date (this returns the number of days between the two dates), dividing by 365.25 to get years, and then applying the int function. The int(*some_number*) function returns the integer portion of *some_number*, that is, it truncates it. We use this since ages are usually rounded down to the nearest integer.

```
if hw_avg ge 90 then hw_grade='A';  
else if hw_avg ge 80 then hw_grade='B';  
else if hw_avg ge 70 then hw_grade='C';  
else if hw_avg ge 60 then hw_grade='D';  
else hw_grade='F';
```

These statements are examples of how to use if-then-else statements in SAS. In general, the statements

```
if a then b;  
else c;
```

mean "if a is true, do b, otherwise do c." The following statements would also do the same thing as the above,

```
if hw_avg ge 90 then hw_grade='A';  
if 80 le hw_avg < 90 then hw_grade='B';  
if 70 le hw_avg < 80 then hw_grade='C';  
if 60 le hw_avg < 70 then hw_grade='D';  
if hw_avg < 60 then hw_grade='F';
```

²² <http://statweb.unc.edu/lgreg/z0245860.htm>

but these are not as efficient, since SAS has to execute all the `if` statements for each observation. In the first set, SAS would execute the `if` statements only until one was true and skip the rest.

```
retain j 0 hwsun1 0 hwsun2 0 hwsun3 0 overall_sum 0;  
j=j+1;
```

Thus far, we have performed computations only within an observation, not across all observations in the dataset. The `retain` statement allows us to do the latter. Normally, when SAS reads in a new observation, it sets the values for all the variables in that observation to missing. Then the values are re-assigned to non-missing values by the statements in the data step. The `retain` statement tells SAS to set the value of a variable to that of the previous observation, instead of setting it to missing.

A simple application of this is calculating the sum of a variable across all observations, which is what we have done in this example. The variables `hwsun1`, `hwsun2`, `hwsun3`, and `overall_sum` (we'll see these shortly) keep running totals of the scores for `hw1`, `hw2`, `hw3`, and all homeworks, respectively. The variable `j` is simply counting the number of observations in the dataset (for use shortly).

The general form of a `retain` statement is:

```
retain var1 initial_value1 var2 initial_value2 etc.;
```

Note: we could have simplified the example statement to:

```
retain j hwsun1-hwsun3 overall_sum 0;
```

since the initial value for the all the variables is zero.

```
array hw[7] hw1-hw3 hwsun1-hwsun3 overall_sum;
```

This statement introduces the use of arrays to simplify programming. An `array` statement is specified as:

```
array array_name[dimension] elements;
```

where *dimension* is the number of elements in the array, and *elements* are variables in your dataset. Note: the variables in the array don't already have to already exist. If you do not include any elements in the array statement, SAS will create the variables `array_name1`, `array_name2`, ..., `array_namen`, where `n` is the dimension of the array. The next set of statements will illustrate the utility of arrays.

```
do i=1 to 3;
  hw[i+3]=hw[i]+hw[i+3];
  hw[7]=hw[7]+hw[i];
end;
```

These statements form a `do` loop, where the first statement is

```
do counter_variable = beg_value to end_value;
```

and the last statement is `end;`. SAS starts with `counter_variable` at `beg_value`. It then reads the statements in the loop and increments `counter_variable` by one. It does this multiple times until `counter_variable` equals `end_value`, and then exits the loop.

In this example, the statements inside the loop involve calling an array, here `hw`, which we specified in the preceding array statement. In this array,

```
hw[1] == the first variable in the array == hw1
hw[2] == the second variable in the array == hw2
etc.
```

So when `i=1`, for example, the loop executes the statements:

```
hwsum1 = hw1 + hwsum1;
overall_sum = overall_sum + hw1;
```

Since the variables `hwsum1`, `hwsum2`, `hwsum3`, and `overall_sum` were retained, this loop calculates the sum of each of the three homeworks for the females in the class and an overall sum of all their homework grades.

Had we not used an array and `do` loop, we would have had to type the following statements (without a `do` loop) to achieve the same result:

```
hwsum1=hw1+hwsum1;
hwsum2=hw2+hwsum2;
hwsum3=hw3+hwsum3;
overall_sum=hw1+hw2+hw3+overall_sum;
```

With such a short `do` loop, using an array and `do` loop doesn't give us much savings in typing, but you can see that for longer `do` loops, savings would be considerable.

```

if eof then do;
  fem_hw1_avg=hwsum1/j;
  fem_hw2_avg=hwsum2/j;
  fem_hw3_avg=hwsum3/j;
  fem_hw_avg=overall_sum/3/j;
end;

```

These statements show how to conditionally execute a block of statements. They take the form

```

if condition then do;
  statements
end;

```

Here, the condition `if eof` is short for `if eof=1`. Since the variable `eof` equals 1 only on the last observation, as described before, this block of statements will be performed only on it and not on the other observations. These statements calculate averages for the females in the class for each of the homeworks and an overall average for all their homeworks, so it makes sense that we would want to perform this calculation only on the last record, since it contains the total sums of the variables `hwsum1`, `hwsum2`, `hwsum3`, and `overall_sum`.

A note about the variable `j`: obviously, we could have easily counted the number of females in the class and just inserted this number wherever `j` appears. But, suppose we want to run this program on the males instead of the females (or on any other subset of the original data). With the variable `j`, we only have to change the `where` statement. Without `j`, we have to physically count how many records apply to the condition in the `where` statement and then change the denominators of each of the statements in the `do` loop to this number. This is a pain; it's much easier just to leave the program more general and use the `j` variable.

```

drop sex dob hw1 hw2 hw3 i j hwsum1 hwsum2 hwsum3
overall_sum;

```

A `drop` statement allows you to delete unneeded variables from your dataset; any variables specified in it will not be written into the new dataset. You could also use a `keep` statement, which works in the reverse way. Here, we could have used the statement:

```

keep name age hw_grade hw_avg fem_hw1_avg
fem_hw2_avg fem_hw3_avg fem_hw_avg;

```

but this takes a little more typing. Notice that the variable `eof` is not included in either of these since it is an internal variable.

Deleting unneeded variables from a dataset can greatly reduce the disk space needed to store it, so it is a good idea to always use `drop` or `keep` statements in your programs.

```
format hw_avg fem_hw1_avg fem_hw2_avg
fem_hw3_avg fem_hw_avg 4.1;
```

A `format` statement applies formats to the variables included in it, as we learned previously.

```
label hw_grade="Homework Grade";
```

A `label` statement attaches a label to a variable in the descriptor portion of the dataset. This is usually something that describes the data the variable contains. Additionally, when printing a dataset, you can specify that labels be printed at the top of your dataset instead of variable names in order to control the appearance of the printout (if a variable doesn't have a label, its name is printed).

The end result of this program is the following dataset, located at `h:\class.sas7bdat`:

Name	Age	Hw_Avg	Hw_Grade	Fem_Hw1_Avg	Fem_Hw2_Avg	Fem_Hw3_Avg	Fem_Hw_Avg
Shannon	19	81.0	B
Sherry	24	94.0	A
Jennifer	18	99.0	A
Mary	21	93.0	A
Kelly	29	97.0	A	93.2	96.8	88.4	92.8

Useful SAS Procedures

Now that you've manipulated your data as you want, you can apply one of the many SAS [procedures](#)²³ to analyze it, using a **proc** step. This section contains an introduction to some of the more commonly used procedures.

The general form of statements in a `proc` step is:

```
proc procedure_name data=dataset_name options;
where some_condition;
by var;
required statement one;
required statement two;
etc.
```

²³ <http://statweb.unc.edu/proc/index.htm>

```
run;
```

The **where** statement is not required, but allows you to subset data for use only in the procedure without having to create a separate data set.

The **by** statement is also not required (except in the `sort` procedure that we'll see shortly) and allows you to perform the procedure on the groups of the variable `var`. For example, the statement `by sex;` would perform the procedure first on the group where `sex='F'` and then on the group where `sex='M'`. To use this statement the dataset must be sorted by the variable `var`.

Label statements can also be used in procedures. When using a `label` statement in a data step, as we saw previously, the label is permanently attached to the variable until you assign the variable a different label. These are used so that when viewing the descriptor portion of the dataset, you can easily see the purpose of each variable. In a procedure, however, a `label` statement assigns a label to a variable only for the duration of that procedure. This is done to control how variables are presented in output. Instead of printing variable names, you can tell SAS to print the variable labels.

In the examples in this section, we'll use the dataset `h:\temp\class.sas7bdat` introduced previously, accessed with the libname `examples`.

Proc Print

The [print](#)²⁴ procedure prints the observations in a dataset in list format, using all or some of the variables. The simplest version of a print step would be:

```
proc print data=dataset_name;
run;
```

We can use options and additional statements to customize the printout. For example:

```
proc print data=examples.class double
    heading=horizontal label;
var name hw1;
sum hw1;
run;
```

- The `var` statement tells which variables to print. Without this statement, all variables will be printed. Note that variables will be printed in the order they are specified in this statement.
- The `sum` statement produces a sum over all observations for the variable(s) specified, here `hw1`.
- The `double=` option writes a blank line between observations.

²⁴ <http://statweb.unc.edu/proc/z0057825.htm>

- The `heading=horizontal` option specifies that variable names will always be printed horizontally. Without this, SAS chooses how to print variable names, according to its own "best" page formatting rules.
- The `label` option specifies that labels be used as variable headings in the printout instead of variable names. If a variable does not have a label attached to it, its name is used by default.

Proc Contents

The [contents](#)²⁵ procedure gives information from the descriptor portion of one or more SAS datasets, such as number of observations, number of variables and their types, lengths, formats, informats, and labels, when the dataset was created and last modified, and what engine was used in creating the dataset. Unlike other procedures, the only statement in the contents procedure is the `proc contents` statement. An example with some useful options is:

```
proc contents data=examples.class out=output_file noprint
    position directory;
run;
```

- The `out=output_file` option allows you to output the information provided by the contents procedure to a file. Additionally you can specify `noprint` to not print anything to the screen.
- By default, the variables in the contents output are ordered alphabetically. To order them by position, use the `position` option.
- To print all the datasets in the same library as the dataset specified in the `data=` option, use the `directory` option.

Proc Sort

A dataset can be sorted by one or more variables using the [sort](#)²⁶ procedure. An example with a few options is:

```
proc sort data=examples.class out=output_file nodupkey
    noequal;
by descending hw3 hw2;
run;
```

- By default, the dataset in the `data=` option is replaced by the sorted version. To create a new dataset `output_file` when sorting, use the `out=output_file` option.
- The `nodupkey` option deletes observations with duplicate values of the `by` variable. Thus, using this option, the sort procedure would produce a

²⁵ <http://statweb.unc.edu/proc/z0085766.htm>

²⁶ <http://statweb.unc.edu/proc/z0057941.htm>

dataset with one observation for each value of the `by` variable(s), here `hw3` and `hw2`.

- By default, SAS maintains the original order of observations within each `by` variable. The `noequal`s option does not necessarily maintain this original order. This option is useful as maintaining the original order takes extra execution time; if you don't need to maintain the order, using the `noequal`s option could make your program could run faster.
- The `by` statement is required and tells SAS which variable(s) to sort by, here `hw3` and `hw2`. By default, observations are sorted in ascending order; you can change this by using `descending` in front of the variable you want to be sorted in descending order, here `hw3`.

Proc Means and Proc Univariate

The [means](#)²⁷ procedure calculates descriptive statistics for numeric variables across all observations or within groups of observations. It also can compute confidence intervals and perform a t-test for the mean. The required statements for this procedure are:

```
proc means data=dataset_name statistics keywords;  
var list of variables;  
run;
```

Some optional statements along with some useful options are included in the following example:

```
proc means data=examples.class median mean std clm  
          probt t noprint;  
class sex;  
output out=stats;  
run;
```

- The options `median`, `mean`, `std`, `clm`, `probt`, and `t` are statistics keywords. The `clm` option calculates a 95% confidence interval for the mean. To specify a different alpha level for this, use the `alpha=` option. The options `probt` and `t` perform a t-test with the hypothesis, `mean = 0`. If no statistics keywords are given, the statistics `n`, `mean`, `std`, `min`, and `max` will be produced.
- The `noprint` option suppresses all printed output.
- The `class` statement works like a `by` statement only the observations do not have to be sorted by the `class` variable, here `sex`. This statement is generally more efficient than a `by` statement and should be used instead if a dataset has not already been sorted.
- The `output` statement outputs descriptive statistics to another dataset, here `stats`.

²⁷ <http://statweb.unc.edu/proc/z0146728.htm>

The [univariate](#)²⁸ procedure does the same things as the `means` procedure, but with more options and more statistics in the default output. It can perform many statistical tests and also can produce crude graphs of the distribution of your data.

Proc Freq

The [freq](#)²⁹ procedure produces frequency tables and counts. It can also perform many tests and calculate measures of association for a particular frequency table. Its syntax is of the form:

```
proc freq data=dataset_name;
  tables list of variables;
  exact keywords;
  test keywords;
  output out=output_file keywords;
run;
```

The `tables` statement is required, whereas, the `exact`, `test`, and `output` statements are not. An example with some options is:

```
proc freq data=examples.class;
  tables hw1*hw2 / all;
  exact chisq;
  test measures;
  output out=stats all;
run;
```

- The `tables hw1*hw2` statement produced a crosstabulation of `hw1` vs `hw2`. Tabulations for single variables can be performed as well. The `all` option (options in statements other than the `data` statement are indicated after a slash) produces all possible table statistics (broken down into `chisq`, `measures`, and `cmh` groups). Different groups of table statistics can be requested here using the appropriate option.
- The statement `exact chisq;` performs exact tests for the `chisq` group of statistics.
- The statement `test measures;` performs asymptotic tests for the `measures` group of statistics.
- Here again, the `output` statement outputs the requested statistics (here `all`) to a dataset, `stats`.

Statistics Procedures

The [SAS/STAT](#)³⁰ product contains over fifty statistical procedures, including [proc reg](#)³¹ (linear regression), [proc glm](#)³² (linear models), [proc logistic](#)³³ (logistic regression), [proc ttest](#)³⁴ (performs t-tests), and [proc anova](#)³⁵ (analysis of variance).

²⁸ <http://statweb.unc.edu/proc/z0146802.htm>

²⁹ <http://statweb.unc.edu/proc/z0146708.htm>

Graphics Procedures

The [SAS/GRAPH](#)³⁶ product contains around twenty procedures that produce various types of charts and graphs. The more commonly used procedures are [proc gchart](#)³⁷, which creates two and three dimensional charts, [proc gplot](#)³⁸, which creates two dimensional plots, and [proc g3d](#)³⁹, which creates three dimensional plots. Additionally, [proc goptions](#)⁴⁰ is used to specify global options pertaining to all graphical procedures.

Global Statements

[Global statements](#)⁴¹ can appear anywhere in a SAS program, either as stand-alone statements or within data or proc steps. One statement like this that we've already seen is the `libname` statement used to specify a particular library. Another is a `comment` statement.

Some other useful global statements are the `page`, `skip`, `footnote`, `title`, `%include`, and `options` statements. We'll discuss each of these individually:

- `page` and `skip` statements can be used to insert extra space between portions of your SAS log, useful in debugging. A `page` statement puts a page break in the SAS log; a `skip` statement inserts a blank line. These are specified with no options, just as the statements `page;` or `skip;`.
- A `footnote` statement adds a footnote at the bottom of every page of printed output. A footnote is specified using the syntax

```
footnote<n> "put your text here";
```

where `n` can range from 1 to 10, meaning you may include up to 10 different footnotes. An example of a footnote giving date and time (useful to identify output) is:

```
footnote1 "Job submitted on &sysdate at &system";
```

³⁰ <http://statweb.unc.edu/stat/index.htm>

³¹ <http://statweb.unc.edu/stat/chap55/index.htm>

³² <http://statweb.unc.edu/stat/chap30/index.htm>

³³ <http://statweb.unc.edu/stat/chap39/index.htm>

³⁴ <http://statweb.unc.edu/stat/chap67/index.htm>

³⁵ <http://statweb.unc.edu/stat/chap17/index.htm>

³⁶ <http://statweb.unc.edu/gref/index.htm>

³⁷ <http://statweb.unc.edu/gref/z0723580.htm>

³⁸ <http://statweb.unc.edu/gref/zlotchap.htm>

³⁹ <http://statweb.unc.edu/gref/zg3dchap.htm>

⁴⁰ <http://statweb.unc.edu/gref/zgopchap.htm>

⁴¹ <http://statweb.unc.edu/lgref/z1225401.htm>

where `&sysdate` and `&systemtime` are internal system variables that will resolve to the current date and time of your SAS session.

- Similar to a `footnote` statement, a `title` statement adds a title at the top of every page of printed output. A title is specified using the syntax

```
title<n> "put your text here" ;
```

where, again, `n` can range from 1 to 10.

- A `%include` statement can be used to include SAS code from another SAS program without actually copying and pasting the code into your current program. This is specified as

```
%include "file location" ;
```

For example:

```
%include "c:\mysaswork\project.sas" ;
```

- The `options` statement allows you to change many [SAS system options](#)⁴². An example of this statement including a few useful options is:

```
options nodate pagesize=56 linesize=80 nonumber  
yearcutoff=1950 ;
```

With the `nodate` and `nonumber` options, the date and page number, respectively, will not appear in the upper right corner of each output page. The `pagesize` and `linesize` options specify the physical size of output pages. The `yearcutoff` option tells SAS how to read two digit dates. As an example, `yearcutoff=1950` means that SAS will interpret two digit dates as between 1950 and 2049. The default for the `yearcutoff` option is 1920. In Windows, options can also be specified from `Tools/Options/System` in the command bar.

Where to Find Help With SAS

New Users

Windows users new to SAS software should definitely look through all the options under the `Help` menu on the command bar. `Getting Started With SAS Software` provides a step-by-step introduction to SAS. The `SAS Online Tutor` is an interactive version of `Getting Started With SAS Software` and includes many sample programs and exercises.

⁴² <http://statweb.unc.edu/lgreg/z0245124.htm>

Reference Information

All SAS documentation is now available in web format in the [SAS Online Documentation](#). This link is available only to campus users. Non-campus users must have the online documentation installed locally and can access it at Help/Books and Training/Online Documentation in the Windows command bar. Other reference information can be found at Help/SAS System Help in the Windows command bar. This contains much more introductory information than the online documentation, but is not as complete for reference purposes.

ATN SAS Documentation

The Applications Support Group at ATN has written a variety of its own [SAS documentation](#)⁴³. This site is updated regularly, so have a look from time to time for interesting new things.

SAS Institute Web Pages

The [SAS Institute website](#)⁴⁴ is a great source for answers to more advanced SAS questions. Good pages to look at are:

- [SAS Notes](#)⁴⁵, the notes SAS technical support personnel reference to answer user questions
- [FAQ's](#)⁴⁶
- [SAS Technical Support Documents](#)⁴⁷
- [Sample SAS Programs](#)⁴⁸

Books

Books and manuals written by SAS personnel or by SAS users can be ordered through [SAS Publishing](#)⁴⁹. The Ram Shop in the Bull's Head Bookstore also sells many of these texts. Additionally, many popular books and manuals are available for check out at the [Applications Support Group Lending Library](#)⁵⁰, located in Phillips 28 (in the basement).

⁴³ <http://help.unc.edu/statistical/applications/sas/>

⁴⁴ <http://www.sas.com>

⁴⁵ <http://www.sas.com/service/techsup/search/sasnotes.html>

⁴⁶ <http://www.sas.com/service/techsup/faq/products.html>

⁴⁷ <http://www.sas.com/service/techsup/tnote/technote.html>

⁴⁸ http://www.sas.com/service/techsup/sample/sample_library.html

⁴⁹ <http://www.sas.com/apps/pubscat/welcome.jsp>

⁵⁰ <http://help.unc.edu/statistical/lendinglib.html>

Help From a Live Person

User support for SAS can be obtained through the [Applications Support Group](#)⁵¹. You can reach us by email at research@unc.edu or by phone at 962-HELP. This phone number takes you to the IT Response Center who will direct your call accordingly. You can also come by and talk to us personally at our office in Phillips 28 (in the basement). We'd be happy to help you with your SAS questions.

⁵¹ <http://help.unc.edu/asg/research/group.html>