

# Syllabus

Lecturer: James Degnan

Office: SMLC 342

Office Hours: M 12:30-1:30, W 1:30-2:30

E-mail: jamdeg@unm.edu

Textbook: Learning SAS by example, by Ron Cody,

[http://unm.worldcat.org/title/learning-sas-by-example-a-programmers-guide/oclc/174000956&referer=brief\\_results](http://unm.worldcat.org/title/learning-sas-by-example-a-programmers-guide/oclc/174000956&referer=brief_results)

Assessment:

1. minor homeworks announced periodically throughout the course (worth 1/4 of grade)
2. Three major homeworks/projects (each worth 1/4 of grade)
3. No tests/exams

The first homework consists of three small problems (1.1, 1.2, 1.3) which are in slides at the end of this set of notes. The goal of the first homework is mostly to make sure that you are getting comfortable running SAS and getting data into the system before doing anything very statistical with it. **It is due Friday, September 7th.**

# Goals of the course

The course focuses on dealing with data in SAS, such as reading in messy data, formatting and reformatting data, and merging data files coming from multiple sources (separate files).

In addition, we will go over macros, which allow you to automate procedures to redo the same analysis for multiple datasets, and the output delivery system, which can extract a small amount of information from output and use this to construct new datasets.

# SAS background

SAS originally stood for Statistical Analysis System and was developed in the 70s from older computer languages. Eventually, it was based on the C language, which is also the basis for other languages still in common use (R, python, perl, etc.)

SAS used to be very commonly used by academic statisticians and is still used in medical research and industry, particularly pharmaceuticals, and in government, such as the FDA and Statistics New Zealand (statistical organization for official statistics in New Zealand).

# SAS background

Other fields might use other software, for example STATA and SPSS are often used in social sciences (sociology, psychology, etc.)

For being a good statistician, especially if you want to consult and collaborate with researchers from other disciplines, it is good to be familiar with more than one software package. This is also a way to check that you understand the syntax and output of different programs (can you get different packages to generate the same output for complex analyses?)

# Advantages of SAS (my opinion)

## Advantages

1. SAS has excellent memory management – can handle data sets with millions of observations without loading everything into memory at the same time
2. SAS is very well documented compared to other programs, so that you can look up what algorithm is used to generate results
3. SAS is relatively stable – there are not new versions being created every six months, making backwards compatibility an issue.
4. SAS is very good at certain aspects of manipulating data, particularly merging data sets and importing standard data files (Excel, Access, etc.)

# Disadvantages of SAS (my opinion)

## Disadvantages

1. SAS is expensive and not intended for individual users (it's intended for large organizations and businesses), making it difficult to get access to.
2. SAS is more difficult to learn than other languages
3. The latest developments might not be available in SAS while they are more likely to be implemented earlier in packages like SAS
4. Graphics are a bit clunky, and easier to do elsewhere

# Accessing SAS at UNM

Good Luck!



# Accessing SAS at UNM

SAS was supported (licenses paid for) at UNM for decades and was available either in specific computer pods or remotely via ssh (for a linux version). Within the last four years, it is no longer supported.

If you work on North Campus, you might be lucky enough to have access. Otherwise, be aware that your professor (James Degnan) also does not have full access. He will instead be using SAS OnDemand for academics.

Many of the slides that will still be used for the course were made using linux version. Many are even in batch mode, which has no GUI and looks quite plain and old-fashioned. These screenshots were made in 2014, before UNM stopped supporting it. Some screen shots may also have been made running SAS from a lab computer, so might look bit different from SAS OnDemand.

# Accessing SAS at UNM

An alternative to SAS OnDemand is SAS University, in which you download stuff to your computer but, as I understand it, still requires an internet connection. You are welcome to try this and share your experience, but the plan will be to mostly use SAS OnDemand.

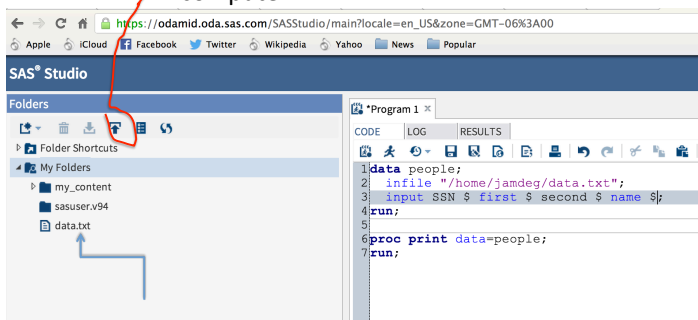
# Other ways to access SAS: SAS OnDemand

SAS OnDemand, which allows you to use SAS online, so that you enter code and upload data, and SAS processes it and runs the code on their servers. You can upload data to their system (up to 10Mb) and download outputted files.

To use this system, follow the instructions in an email I will send to you titled SAS OnDemand.

# SAS OnDemand

Upload data from your computer

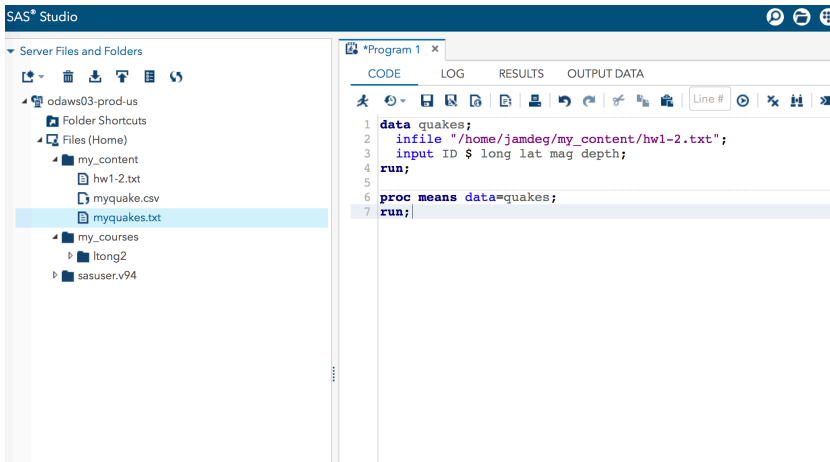


The screenshot shows the SAS Studio web interface. The browser address bar displays [https://odamid.oda.sas.com/SASstudio/main?locale=en\\_US&zone=GMT-06%3A00](https://odamid.oda.sas.com/SASstudio/main?locale=en_US&zone=GMT-06%3A00). The interface has a dark blue header with the 'SAS® Studio' logo. Below the header, there's a 'Folders' pane on the left and a 'CODE' pane on the right. The 'Folders' pane shows a tree structure under 'My Folders' with subfolders 'my\_content' and 'sasuser.v94', and a file 'data.txt'. The 'CODE' pane shows a SAS program with the following code:

```
1 data people;  
2   infile "/home/jamdeg/data.txt";  
3   input SSN $ first $ second $ name $;  
4 run;  
5  
6 proc print data=people;  
7 run;
```

If data set uploads, it should be listed here.

# SAS OnDemand



The screenshot displays the SAS Studio web interface. On the left, the 'Server Files and Folders' pane shows a tree structure under 'odaws03-prod-us'. The 'Files (Home)' section is expanded, showing a folder named 'my\_content' which contains 'hw1-2.txt', 'myquake.csv', and 'myquakes.txt'. The 'myquakes.txt' file is selected and highlighted in light blue. Below it are other folders like 'my\_courses' and 'sasuser.v94'. On the right, the 'CODE' tab is active for '\*Program 1'. The code editor shows the following SAS code:

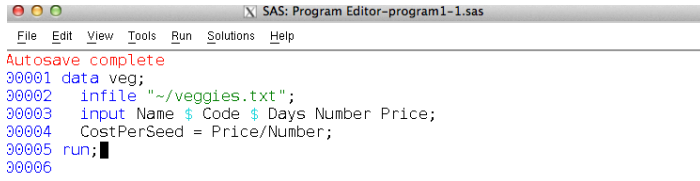
```
1 data quakes;  
2   infile "/home/jamdeg/my_content/hw1-2.txt";  
3   input ID $ long lat mag depth;  
4   run;  
5  
6 proc means data=quakes;  
7 run;
```

The code editor includes a toolbar with icons for running, saving, and other file operations, and a 'Line #' input field.

## Other ways to access SAS: SAS University

This allows you to download an educational version of SAS, which I believe is SAS Studio. The download might be a bit complicated, and I might have difficulty helping (I couldn't get it to work on my computer). SAS is 1.6G to download. You have to have virtualization software (over 100Mb) and VMware Player or VMware Fusion (500 Mb for me), which has a 30-day free trial and is otherwise 60 dollars. Instructions are at [http://www.sas.com/en\\_us/software/university-edition/download-software.html](http://www.sas.com/en_us/software/university-edition/download-software.html)

# Example program from book, page 5

A screenshot of the SAS Program Editor window. The title bar reads "SAS: Program Editor-program1-1.sas". The menu bar includes "File", "Edit", "View", "Tools", "Run", "Solutions", and "Help". The text area contains the following SAS code:

```
Autosave complete
00001 data veg;
00002   infile "~/veggies.txt";
00003   input Name $ Code $ Days Number Price;
00004   CostPerSeed = Price/Number;
00005 run;
00006
```

Note that in linux/unix, directories are separated by forward slashes rather than backward slashes. The `~` indicates the home directory in linux. Windows uses backward slashes. A typical Windows `infile` statement might be something like

```
infile "c:/CLASS/SAS/veggies.txt";
```

# SAS programs

SAS programs can be written directly into the Program Editor (one of the windows that pops up when you first load SAS) or SAS Display Manager (appears to not be available in the linux version). This is initially very small, so you might want to expand it. You might want to save your work to a file before running to save your work. SAS files with program code can be opened from the Program Editor using File->Open from the menus. To run code, go to the menu bar in the Program Editor, and click on Run->Submit.

It is also possible in Windows to highlight code or copy and paste code from the clipboard to run a segment of code instead of running the whole program.



# Basics of SAS syntax

1. SAS is usually not case sensitive, except for file names and data, but will preserve cases as entered by the user, e.g., `data=veg` , `data=VEG`, and `DaTa = VEG` are all equivalent, but in output, SAS will preserve the case first used when outputting results from the dataset called `veg`
2. whitespace is usually not important
3. lines are ended by semi-colons, not whitespace or line returns
4. Variable names chosen by the user must start with letters or underscores (but not blanks or other special characters), and can be up to 32 characters. (Older versions of SAS had a limitation of 8 characters.) Numbers can be including if they are not the initial character.
5. Variables can be either numeric or character (no distinction between integer and numeric), and are 8 bytes (roughly 14 digits of precision for numeric)

# Basics of SAS syntax

Programs typically have a combination of DATA steps and PROCs. DATA steps read-in and manipulate data, such as creating new variables that are functions of other variables. PROCs often do statistical procedures, sorting, reformatting, etc.

DATA steps begin with `data NAME` where NAME is the name of the data set being created. PROCs begin with `proc` followed by the name of the procedure, e.g., `proc means`, `proc sort`, `proc regress` etc. Statistical functions such as linear models are handled by PROCs.

DATA steps and PROCs are evaluated when there are `run` statements. Typically, you should have one `run` at the end of each DATA step and each PROC.

# DATA steps

Much of this course will be based on the DATA step. With the DATA step, you can either read in data from a file, write the data directly into your SAS code (useful only for small examples), or read in data from other datasets in SAS memory (could be read in earlier in your program).

Sometimes it is useful to have two versions of the dataset, maybe one with outliers removed, and one with the original data, or just a subset of cases.

# Multiple datasets

You can also do things like reorganize a dataset so that it has say, one record (i.e., row) per hospital visit versus one record per patient (with multiple hospital visits per patient). Similarly for repeated measures data, you might one record per individual or one record per combination of individual and time point depending on the statistical procedure you want to use later on.

Another important function is merging data sets. You might have one data set from a hospital and another from an HMO and you need to combine this into one data set with one record per patient where the same patient might have records in both the hospital and HMO data. The two data sets might have overlapping but not identical fields (variables), which you don't want duplicated.

# Options and global statements

There can also be statements that are outside of DATA steps and PROCs which are called global statements.

Examples are options, which can do things like control the width of the output and whether or not page numbers or the date are displayed. In addition, title statements can be used, which can help decipher output. Later in the course, we will use macros which will allow us to loop over DATA steps and PROCs. Another example is comments. There are two ways to have comments. For example,

```
data mydata;  
*The following nested comment is false;  
/* *The previous comment is false; */  
    infile "~/mydata.txt";  
    input var1 var2 * var3; ;  
    /* input var4 var5 var6; */  
run;
```

# Comments on comments

As a general comment, I will not require comments in your homework, but be aware that other instructors might require them for other courses. Comments are good if they help you.

In my experience, when I have read other people's code for academic research, there are usually no comments. Examples include SAS macros that you might download from the web or R code, which is pen source, but usually not commented. The larger the code, the more important the comments are. Comments can help you if you need to examine the code later (e.g., 6 months later or 6 years later) or want someone else to modify your code, so are often a good idea.

On the other hand, they can slow you down, and if you change code somewhere, you might have to change comments elsewhere in the program, making comments inaccurate if they are not checked.

# Reading in data

1. Typically, data is read in from the `infile` line, which specifies the file name for one data set.
2. The `infile` statement is typically followed by an `input` statement which gives names to the variables.
3. The default is for variables to be numeric. Character variables are indicated by \$.

```
data veg;  
infile "c:\books\learning\veggies.txt";  
input Name $ Code $ Days Number Price;  
run;
```

Here `Name` and `Code` are character while the other variables are numeric. Note that a variable which has only numerals can be specified to be character (e.g., 0 and 1 for male and female).

# Reading in data

Often values in plain text files are separated by white space, which might or might not line up neatly into columns. Different columns (variables) might be organized by being separated by an arbitrary number of spaces, by tabs, or by some other character, such as commas.

A convenient way to read in Excel spreadsheets is to save them as Comma Separated Values (csv) files, which are plain text. Comma separated files are useful when a field might have text that includes whitespace, such as names of individuals, streets, cities. For example, how many columns (variables) are in the following data?

```
Lily Rose West Mesa Vista Ave NE Albuquerque  
Willard van Orman Quine Cactus Circle NW Rio Rancho  
Jose Ortega y Gasset East Desert Willow Rd Phoenix
```



## Reading in data: comma separated values

The desired variables here are not clear. Is the name one long string? Is it desirable to have separate first and last names? Should the middle name be separated? Is NE part of the road name? Is it missing for addresses in Arizona? The following makes it a bit clearer

```
Lily Rose,,West,Mesa Vista Ave NE,Albuquerque  
Willard,van Orman,Quine,Cactus Circle NW,Rio Rancho  
Jose,,Ortega y Gasset,East Desert Willow Rd,,Phoenix
```

Note the use of the double commas. Two commas in a row means that the variable is missing for that observation (row).

# Missing values

Real-world datasets often have missing values. There are many ways that missing values can be coded in a dataset, and you need to have ways of dealing with these when you read in data to any software package.

Here are some common ways to specify a missing value

1. a blank space (usually not a good idea – you might fix the file before inputting into SAS)
2. a period: .
3. an expression like NA (not available) or NaN (not a number)
4. a code such as -99 for positive integer data
5. using a 0 (this can be misleading if read as a numeric value – might convert file before inputting in to SAS)
6. two commas (or other delimiters, such as a dollar sign or semi-colon) in a row for delimited data with no space in between

# Examples of missing values

```
12 45 13 .  
. -11 12 14  
23 . . .
```

or

```
12 45 13 NaN  
NaN -11 12 14  
23 NaN NaN NaN
```

For now, we will assume that missing values are handled either by using a delimiter or by using a '.'. Different procedures might work differently with missing data. For example, an average for a variable might be computed using nonmissing values.

## Example with missing data

Consider the following data. We have text file that stores the data. The data is read in SAS and means for each variable are computed using PROC MEANS. The variables here are EarthquakeID, longitude, latitude, magnitude, and depth for earthquakes in New Zealand (after 8pm on March 31st, 2011). Depth is in kilometers

```
3489855 166.77054 -45.47349 3.016 12
3489852 172.71054 -43.585 2.937 6.9356
3489806 172.70467 -43.48363 2.338 .
```

How can you tell how many observations were missing from the output of PROC MEANS? Repeat this exercise using a blank instead of a period '.' for the missing value.

# Missing data example: submitting a SAS program

The screenshot shows the SAS Program Editor window titled "SAS: Program Editor-Untitled". The menu bar includes File, Edit, View, Tools, Run, Solutions, and Help. The editor contains the following SAS code:

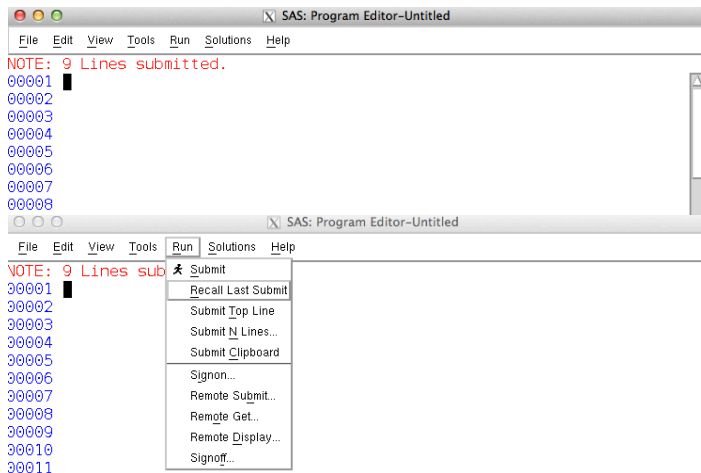
```
NOTE: 9 Line(s) recalled.
00001 data quakes;
00002   infile "Teaching/SAS/hw1-2.txt";
00003   input ID $ longitude latitude magnitude depth;
00004 run;
00005
00006 proc means data=quakes;
00007 run;
00008
00009 options nodate;
00010
00011
```

The "Run" menu is open, displaying the following options:

- Submit
- Recall Last Submit
- Submit Top Line
- Submit N Lines...
- Submit Clipboard
- Signon...
- Remote Submit...

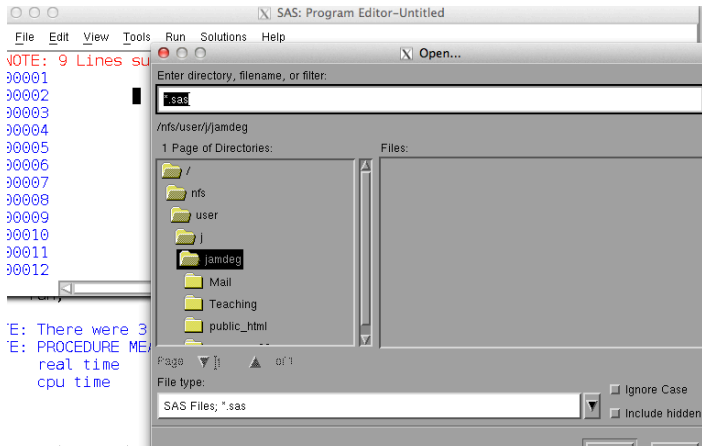
# Missing data example: getting the code back

Click on Run->Recall Last Submit



# Missing data example: getting the code back

Alternatively, you can reload the program if it is saved as a .sas file, using File->Open and navigating through directories to your SAS file.

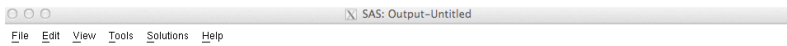


## Missing data example: output

When PROC MEANS is run, the output window will be made visible (assuming everything worked), and you can see the results. Recall that this example shows the difference in output depending on whether the missing value was coded as a period versus blank.



First example uses period for missing value, second example has blank.



### The SAS System

#### The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
longitude	3	170.7285833	3.4277673	166.7705400	172.7105400
latitude	3	-44.1807067	1.1207299	-45.4734900	-43.4836300
magnitude	3	2.7636667	0.3707483	2.3380000	3.0160000
depth	2	9.4678000	3.5810716	6.9356000	12.0000000

■

#### The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
longitude	2	169.7405400	4.2002143	166.7705400	172.7105400
latitude	2	-44.5292450	1.3353641	-45.4734900	-43.5850000
magnitude	2	2.9765000	0.0558614	2.9370000	3.0160000
depth	2	9.4678000	3.5810716	6.9356000	12.0000000

■

# Missing data in SAS

There is often more than one way to tell if your dataset has missing values. One way here is from the output, which indicates the sample size from the column for  $N$  (although  $N$  was not a variable). You can also check by looking at the log output that SAS generates.

It is a good habit to look at the log file to make sure that you are correctly analyzing and reading in your data. It is generally a good idea to check that the number of observations is correct based on your original file, and to check whether any data is missing that shouldn't be. Other output can be useful for diagnosing problems. For example, checking that the Minimum and Maximum values make sense for the range of your data make sense (e.g., latitudes should be negative for New Zealand, magnitudes of earthquakes must be between 0 and 10, etc.).

# Switching between windows in SAS

SAS has a lot of separate windows including the Program Editor, Output, Log, Results, and Graphics (if graphics are generated) window. To switch back and forth between Windows, you can go to View from the menu bar for a window and switch to another window. So you might want to switch between Output, Log, and Program windows. You can also do this using key combinations such as Command+left quote on a Mac and probability ALT+left quote on PCs.

The SAS System

The MEANS Procedure

	N	Mean	Std Dev	Minimum	Maximum
magnitude	3	170.7285833	3.4277673	166.7705400	172.7105400
depth	3	-44.1807067	1.1207299	-45.4734900	-43.4836300
	3	2.7636667	0.3707483	2.3380000	3.0160000
	2	9.4678000	3.5810716	6.9356000	12.0000000

## Missing data example: log file

Here we'll look at part of the log for the missing data example using the period) to see what it says. The first example is with the period used for missing data. Missing data is not indicated in the log file, but it tells us the number of observations and number of variables. The line numbers next to the code is due to having previously run code before, and the log file is appended to during your session. You can clear the log file (making it less confusing to read) by going to Edit->Clear All..

```
96 data quakes;  
97     infile "~/Teaching/SAS/hw1-2.txt";  
98     input ID $ longitude latitude magnitude depth;  
99     run;
```

NOTE: The infile "~/Teaching/SAS/hw1-2.txt" is:  
Filename=/nfs/user/j/jamdeg/Teaching/SAS/hw1-2.txt,  
Owner Name=jamdeg,Group Name=academic,  
Access Permission=rw-----,  
Last Modified=Wed Aug 13 15:10:55 2014,  
File Size (bytes)=112

NOTE: 3 records were read from the infile "~/Teaching/SAS/hw1-2.txt".  
The minimum record length was 35.  
The maximum record length was 38. ■

NOTE: The data set WORK.QUAKES has 3 observations and 5 variables.

## Missing data example: log file

Now we see the logfile when the missing data is an empty space. Note that it says that there is a lost card, meaning one observation is missing (from the days when data were entered into on paper punch cards!), and the entire third row of data is lost. Note also that SAS says that it created a SAS dataset called WORK.QUAKES.

```

103
104 options nodate;
105 data quakes;
106   infile "~/Teaching/SAS/hw1-2.txt";
107   input ID $ longitude latitude magnitude depth;
108 run;

```

NOTE: The infile "~/Teaching/SAS/hw1-2.txt" is:  
 Filename=/nfs/user/j/jamdeg/Teaching/SAS/hw1-2.txt,  
 Owner Name=jamdeg, Group Name=academic,  
 Access Permission=rw-----,  
 Last Modified=Wed Aug 13 15:47:02 2014,  
 File Size (bytes)=111

NOTE: LOST CARD.

ID=3489806 longitude=172.70467 latitude=-43.48363 magnitude=2.338 depth=. \_ERROR\_=1 \_N\_=3

NOTE: 3 records were read from the infile "~/Teaching/SAS/hw1-2.txt".

The minimum record length was 34.

The maximum record length was 38.

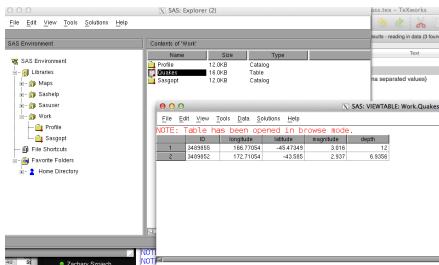
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.

NOTE: The data set WORK.QUAKES has 2 observations and 5 variables.



## Missing data example: SAS dataset

To see the SAS dataset go to View->Explorer. This includes some internal objects in SAS, including Work, which is where temporary datasets are stored. If you try double clicking on Work, you should see Quakes listed, including its size and the type of object it is. If you double click on Quakes here, it will open up Quakes in a spreadsheet-like format, which can be much easier to visually inspect than the original text file. This matters more for large files with many variables.



## Delimited data: comma delimited data

As mentioned previously, data fields can be delimited by commas or other characters. To read a comma delimited file in SAS, use the following

```
infile "~/file.txt" dsd;
```

Which tells SAS that the file has delimiter-sensitive data. The default value for a delimiter is a comma, the delimiter used for .csv files.

## Delimited data: user-defined delimiters

As a user, you can specify your own delimiter if you don't want a comma. For example, income data might have commas in the middle of a dollar figure, so instead of a comma, you might use another character, such as a semi-colon or colon.

To use another character, you can use

```
infile "~/file.txt" dlm = ";";
```

To use a semi-colon as a delimiter.

## Delimited data: strings as delimiters

Sometimes you might want to use more than one character as a delimiter. For example, in the address example above, if every field is separated by two spaces, and strings have at most one space within them, then you can tell whether a bit of text belongs to a person's name rather than the street by the number of spaces.

There are ways around this instead of reading directly into SAS. For example, you could do a global copy and replace in Word or other program to replace double spaces with semi-colons. Then if there were more than two spaces in a row separating some fields, you'd end up with some double semi-colons, which should only be single semi-colons. Another global copy and replace, replacing double semi-colons with single semi-colons would be needed (maybe several times).

But you can also read this type of data into SAS directly, although it is a bit tricky.

# Delimited data: strings as delimiters

In this example, we just want the variables NAME, ADDRESS, CITY. We use `dlimstr` for a string delimiter instead of a single character delimiter

Lily Rose West	Mesa Vista Ave NE	Albuquerque
Willard van Orman Quine	Cactus Circle NW	Rio Rancho
Jose Ortega y Gasset	East Desert Willow Rd	Phoenix

Code:

```
data address;
  infile "~/Teaching/SAS/address.txt" dlimstr=" ";
  input name :$41. address :$41. city :$41.;
run;

proc print data=address;
run;
```

## Tab delimited data

Sometimes data fields are separated by tabs rather than spaces. This is usually not clear from looking at a file, but if you use the cursor in the file and it suddenly jumps several spaces when moving the arrow key one space, it might be tab-delimited.

Although tabs look like multiple spaces, they are represented differently in the computer. Internally, each character is represented by a numerical code, such as ASCII for plain text files, and a tab has a different code than spaces or multiple spaces.

To read in tab-delimited data, you can specify the ASCII hexadecimal code as the delimiter in most cases. Here you use (for example)

```
infile "~/myfile.txt" dlm = "09"x
```

## Other ways to read in data: column input

In addition to having data delimited by spaces, tabs, or other characters, you can read in data assuming that it starts in a certain column.

This is only useful if your data is very structured, but can be useful way to re-order columns or not read in all of the data. It can also be a way to extract only the information needed.



## Line input example

Example, suppose this is data.txt which has social security numbers (made up), name, and date-of-birth. Suppose only the last four digits of the SSN number and last two digits of birth year are needed.

555-91-0344	Lily Rose West	06-23-1966
554-99-0354	Willard van Orman Quine	06-25-1908
344-05-0226	Jose Ortega y Gasset	05-18-1983

SAS program (assuming that I have the columns right – hard to get right for the slide...). What does this do?

```
data people;  
  infile "~/Teaching/SAS/data.txt" ;  
  input  year1 57-58 year2 55-58 SSN $ 8-11;  
run;
```

# Reading data within the SAS program instead of from an external file

Instead of reading in data from an external file, you can do it directly in the program using `datalines`, which is useful to check quick problems and for small datasets. It is more important to know how to read in data from a file, however, since this is much more common. The following is an example:

```
data demographics;
infile datalines dsd;
input Gender $ Age Height Weight;
datalines;
"M",50,68,155
"F",23,60,101
"M",65,72,220
"F",35,65,133
"M",15,71,166
;
run;
```

# Homework problem 1.1

Consider the dataset

```
Cucumber 50104-A 55 30 195
Cucumber 51789-A 56 30 225
Carrot 50179-A 68 1500 395
Carrot 50872-A 65 1500 225
Corn 57224-A 75 200 295
Corn 62471-A 80 200 395
Corn 57828-A 66 200 295
Eggplant 52233-A 70 30 225
```

Now consider replacing the observation

Corn 57828-A 66 200 295 with corn 57828-A 66 200 295. Does this change the output of the following code? If so, how?

```
title "Frequency Distribution of Vegetable Names";
proc freq data=veg;
tables Name;
run;
```

## Data for HW 1.2, slide repeated from earlier

In this example, we just want the variables NAME, ADDRESS, CITY. We use `dlimstr` for a string delimiter instead of a single character delimiter

Lily Rose West	Mesa Vista Ave NE	Albuquerque
Willard van Orman Quine	Cactus Circle NW	Rio Rancho
Jose Ortega y Gasset	East Desert Willow Rd	Phoenix

Code:

```
data address;  
  infile "~/Teaching/SAS/address.txt" dlimstr="  ";  
  input name :$41. address :$41. city :$41.;  
run;  
  
proc print data=address;  
run;
```

## Homework 1.2

Consider the code on the previous slide. Modify the `infile` line so that this works on your system. You will need to create the file `address.txt` yourself using (for example) Notepad or `pico` in LINUX. Describe what happens (by trying it out) with the following variations:

1. you remove the colons before the dollar signs in the input line
2. you remove the periods in the input line, i.e., you type `$41` instead of `$41.`
3. you use `dlim=" "` instead of `dlimstr=" "`
4. you use `dmlstr=" "` instead of `dlimstr=" "` (one space instead of two)
5. you remove `41.` after the dollar signs in the input line
6. you replace `41.` with `1.` in the input line

## Homework 1.3

Create a file with data and write a SAS program to read in the data below and create a new variable `time`, which computes the amount of time in years between the first and last visits for the people listed (compute time ignoring months and dates, so that the time between 06-14-85 and 02-29-2000 is  $2000 - 1985 = 15$ ). Use `proc print` to print the new data. Make sure that none of the information in the data is missing when the new dataset is printed. Note that this data was read in and printed earlier in the slides (you'll have to look for where!), but was read in incorrectly, resulting in some information being lost. Note that years are recorded with either 2 digits or 4 digits depending on the variable.

```
556-45-9565 06-13-85 07-14-2006 david  
556-65-7687 06-13-85 07-31-2013 liang  
556-33-4325 06-14-85 02-29-2000 beatrice  
575-56-3322 06-14-85 01-31-2014 jenny
```

# Reading in data with formats

Often you want to format data a certain way when it is read in. For example, you might want to only read in numeric values to two decimal places, and get SAS to display dollar amounts with dollar signs and values like 1335 as 1,335. Dates also have a special format.

```
data list;  
    informat DOB mmddyy10.  
                Salary dollar8.;  
infile datalines;  
input $ DOB sex $ Salary  
format DOB date9. Salary dollar8.;  
datalines;  
10/21/1955 M 1145  
11/18/2001 F 18722  
05/07/1944 M 123.45  
07/25/1945 F -12345  
;  
run;  
title "debug 2";  
proc print data=list; run;
```



Note that there are MANY SAS date formats. Dates are converted into an integer that is a value that represents the number of days since January 1, 1960, with January 1st having a date of 0.

“SAS can perform calculations on dates ranging from A.D. 1582 to A.D. 19,900. Dates before January 1, 1960, are negative numbers; dates after are positive numbers.” from

<https://v8doc.sas.com/sashtml/lrcon/zenid-63.htm>

We'll get into using formats in more detail later in the semester. You don't need date formats for the first homework.

# SAS date formats

List	Input	Result
DATEw.	14686	17MAR00
DATE9.	14686	17MAR2000a
DAYw.	14686	17
DDMMYYw.	14686	17/03/00
DDMMYY10.	14686	17/03/2000
DDMMYYBw.	14686	17 03 00
DDMMYYB10.	14686	17 03 2000
DDMMYYCw.	14686	17:03:20
DDMMYYC10.	14686	17:03:2000
DDMMYYDw.	14686	17-03-00
DDMMYYD10.	14686	17-03-2000
DDMMYYNw.	14686	17MAR00
DDMMYYN10.	14686	17MAR2000
DDMMYYPw.	14686	17.03.00
DDMMYYP10.	14686	17.03.2000
DDMMYYSw.	14686	17/03/00
DDMMYYs10.	14686	17/03/2000
DOWNAME.	14686	Friday