Importing a wide range of data formats

Steve Simon

7/1/2019

Importing choices (1 of 2)

- A wide range of formats
  - Space delimited
  - Comma separated values
  - Tab delimited
  - Fixed format
- Strings in your input

I want to show a few simple examples of importing data. There are several common formats and SAS can handle all of these easily.

Importing choices (2 of 2)

- proc import
  - First row includes variable names
  - Binary data files
- Manual reformatting
  - Global search and replace
  - Not usually a good idea
- Skipping rows
- Converting strings to numbers

Some data sets use the first row to represent variable names while others don't.

If you have to, you can manually reformat the data. Use the global search and replace function in your text editor program. I generally try to avoid this. If your data set changes, you have to redo the reformatting step, which is tedious and error prone. You'd be better off learning a few tricks to get SAS to read a nonstandard data set.

Sometimes you have to skip a few rows. Sometimes you have to convert strings to numbers.

Space delimited, example

```
4 8 40
8 16 80
12 24 120
16 32 160
24 48 240
```

You've already imported a file similar to this. It is a space delimited file with one blank between each number.

Space delimited, SAS code

```
filename raw_data "../data/space-delimited.txt";
libname module01 "../data";

data module01.space_delimited;
  infile raw_data delimiter=" ";
  input x y z;
run;

proc print
    data=module01.space_delimited(obs=2);
  title1 "First two rows of data";
run;
```

Here is the SAS code to read and print this file.

Multi-space delimited, example

```
 4  8  40
 8 16  80
12 24 120
16 32 160
24 48 240
```

Some text files will insert extra spaces, as in the first two lines of this data set, to align the data. It makes the file more readable. SAS will handle multiple blanks, including blanks at the beginning and end of a line, in a way that you probably will want.

Note that multiple blanks as a delimiter is not a problem, but try to avoid this for other delimiters, especially commas.

Multi-space delimited, SAS code

```
filename raw_data "../data/multi-delimited.txt";
libname module01 "../data";

data module01.multi_delimited;
  infile raw_data delimiter=" ";
  input x y z;
run;

proc print
    data=module01.multi_delimited(obs=2);
  title1 "First two rows of data";
run;
```

Here is the SAS code to read and print this file. Nothing has changed except the name of the raw data (in the filename statement, not shown) and the sas data set name.

Comma delimited, example

```
4,8,40
8,16,80
12,24,120
16,32,160
24,48,240
```

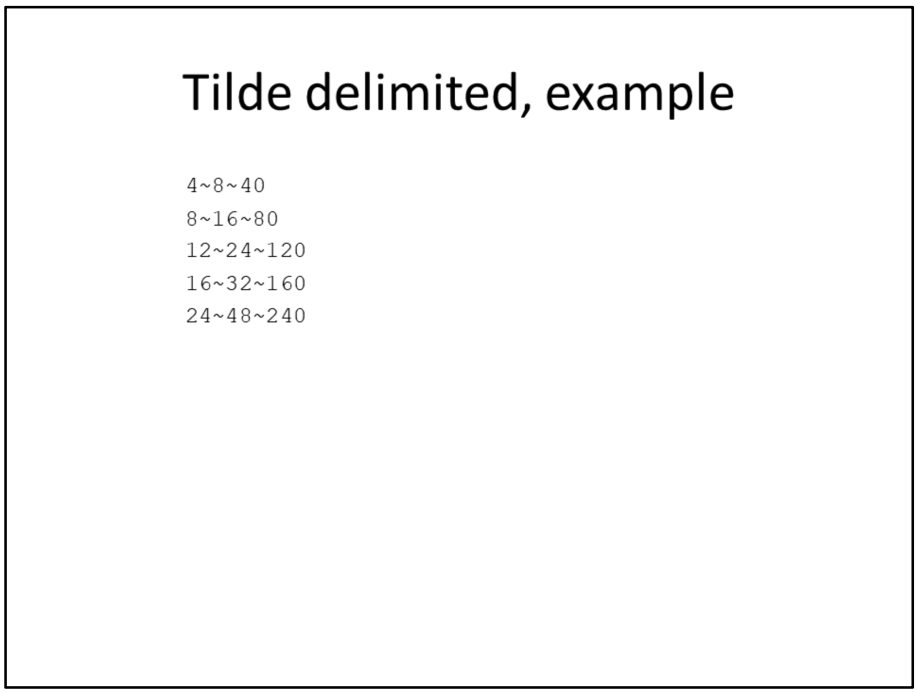
Many text files use commas to separate individual numbers. These often use a special file extension, .csv. These files are especially popular with Excel users, as they are about as easy to import and export as any other type of simple text files.

Here's a simple example of a comma delimited file.

Comma delimited, SAS code

```
filename raw_data
  "../data/comma-delimited.csv";
libname module01
  "../data";
data module01.comma_delimited;
  infile raw_data delimiter=",";
  input x y z;
run;

proc print
    data=module01.comma_delimited(obs=2);
  title1 "First two rows of data";
run;
```

Here is the SAS code to read and print this file. Nothing has changed other than the name of the file and the delimiter option on the infile statement.



You might even see a bizzare symbol as a delimiter. Something that almost never appears. The tilde(~) is a good example. We have not talked about string data yet, but if your data has lots of strings and it has a lot of unmatched quote marks, commas, and other weird things that might trip you up, the owner of the data might use a tilde because nothing else works as a delimiter.

Here's a simple example of a tilde delimited file.

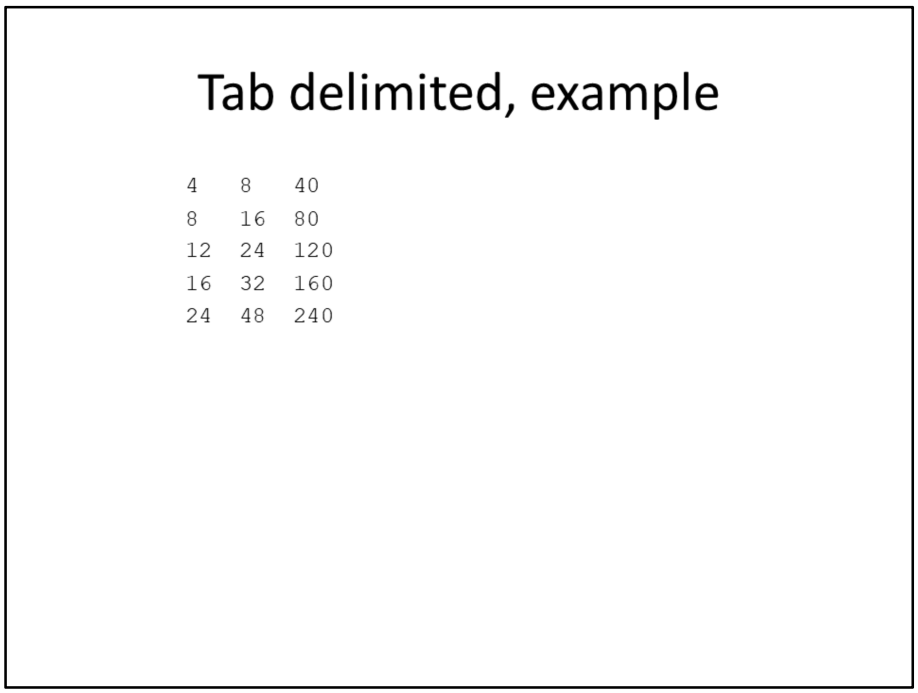
Tilde delimited, SAS code

```
ods pdf file="import-tilde-delimited.pdf";
filename raw_data
  "../data/tilde-delimited.txt";
libname module01
  "../data";
data module01.tilde_delimited;
  infile raw_data delimiter="~";
  input x y z;
run;

proc print
    data=module01.tilde_delimited(obs=2);
  title1 "First two rows of data";
run;
ods pdf close;
```

Here is the SAS code to read and print this file. Again, nothing has changed except the file names and the delimiter option on the infile statement.

Break #1

- What have you learned.
  - Space delimited files
  - Comma separated values
  - Tilde delimited files
- What's coming next
  - Tab delimited files
  - Fixed width files
  - First line variable names
  - Converting strings to numeric



I've already told you that tabs are evil, but if you encounter a tab delimited file, don't panic. You can handle it.

Here's a simple example of a tab delimited file. Note that the numbers are left justfied, which is a hint that there are tabs lurking in the file. The tabs here stop at columns, 5, 9, and 13, which is a bit weird, but may just represent how my system treats tabs. The classic thing to look for in a tab delimited file, besides the right justification, is the semi-aligned, but not perfectly aligned numbers. This doesn't happen with this data because all of the numbers are three digits or less, but it can happen when the numbers take up a bit more room and the numbers are uneven in size.

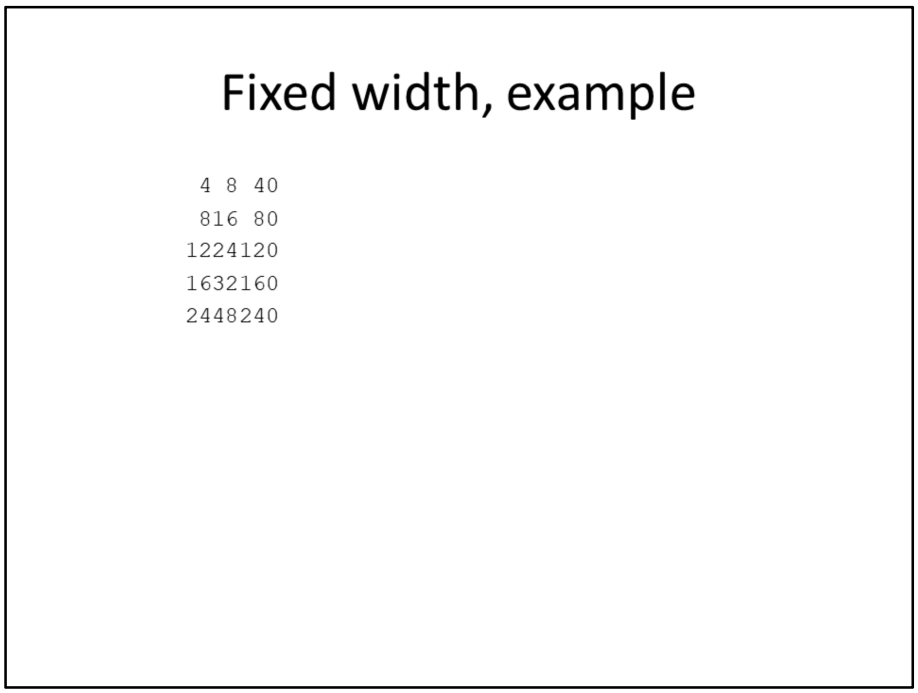
You can also detect tabs by opening the file in a text editor like notebook and playing around with adding extra spaces. A lack of action followed by a sudden hop is a pretty good indication that you are dealing with tabs.

Tab delimited, SAS code

```
filename raw_data
  "../data/tab-delimited.txt";
libname module01
  "../data";
data module01.tab_delimited;
  infile raw_data delimiter="09"X;
  input x y z;
run;

proc print
    data=module01.tab_delimited(obs=2);
  title1 "First two rows of data";
run;
```

Here is the SAS code to read and print this file. You cannot enter the tab character directly as a delimiter. Instead, you have to provide the hex code, 09. Note the use of quote marks and X.

This import worked fine, but beware. Some of these tab separated value files have a few stray blanks mixed in with the tabs, and this can cause havoc.



Sometimes you may run across a file with no delimiters at all. The numbers stand next to each other shoulder to shoulder. Here's an example. There's a space or two here or there because not every number is exactly the same number of digits. But there are plenty of numbers without any spaces behind them.

Why might you do this? If your file was very large, you may not want to afford the luxury of delimiters. The file is already straining to fit in and adding delimiters would just be too much.

These days, of course, storage is cheap, but you still might encounter a fixed width file now and then.

The only way an undelimited file would work is if there was a rigid structure where each number would reside in a specified column location. It's like a series of row houses and the address tells you which house is which.

The key that tells you what columns correspond to which numbers has to reside in a different file. And you have to translate this key to the proper SAS code.

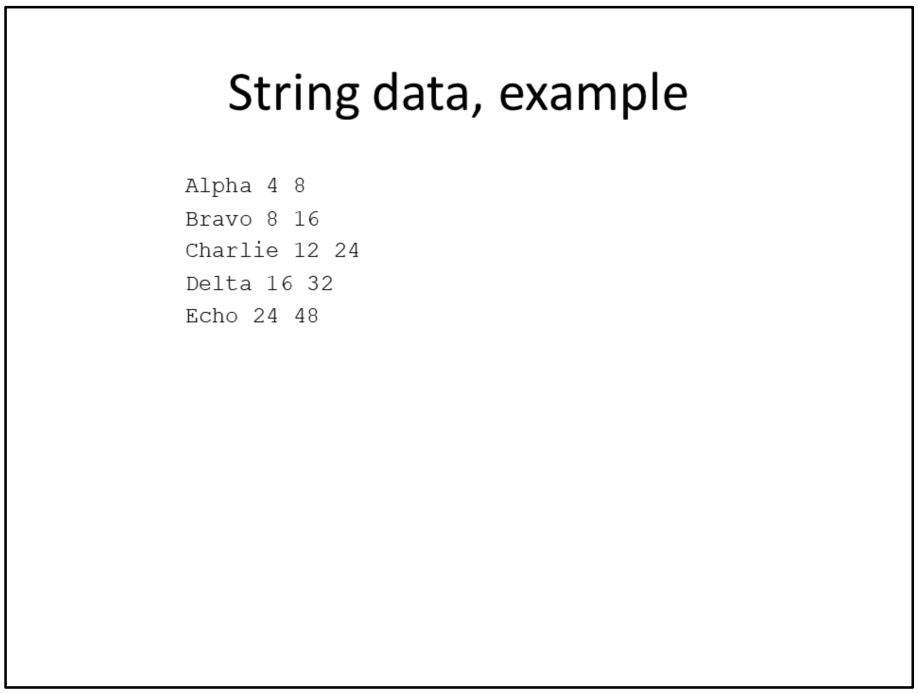
In the fixed width file above, you would have to be told there are three numbers, the first in

columns 1 and 2, the second in columns 3 and 4 and the last in columns 5 through 7.

Fixed width, SAS code

```
filename raw_data
  "../data/fixed-width.txt";
libname module01
  "../data";
data module01.fixed_width;
  infile raw_data delimiter=",";
  input
    x 1-2
    y 3-4
    z 5-7;
run;

proc print
    data=module01.fixed_width(obs=2);
  title1 "First two rows of data";
run;
```

Here is the SAS code to read and print this file. Each variable is followed by a column range.



Strings add a bit of complication. A string is a sequence of one or more letters, like a person's name, or a mixture of numbers and letters, like a person's address.

Here's a small text file with a string of four to six letters.

String data, SAS code

```
filename raw_data
  "../data/string-data.txt";
libname module01
  "../data";
data module01.string_data;
  infile raw_data delimiter=" ";
  input
    name $
    x
    y;
run;

proc print
    data=module01.string_data(obs=2);
  title1 "First two rows of data";
run;
```

Here is the SAS code to read and print this file. Designate a string variable by adding a dollar sign after the variable name.

Complications with string data

- Strings longer than eight characters
  - Informat statement
- Strings with delimiters
  - Use different delimiter
  - Enclose sting in quotes
- Strings with quotes
  - Use double quotes around string with single quote
    - "It's my bidthday!"
  - Use single quotes around string with double quote
    - 'Smile and say "Cheese!" when I take this picture'
  - Use escape codes

SAS will have problems with certain strings. I don't want to go into too much detail now, but here are some things to look out for.

SAS expects strings to be eight characters in length or less. If your string has more than eight character, you have to warn SAS with an informat statement.

If you have a space delimited file, but your strings have space in them, the simplest thing is to switch to a different delimiter, like a comma. What if your string has commas and spaces? Well you could try using a tilde, or some other obscure character. Another choice is to enclose your string in quotes.

What about strings with quotes in them. This can happen. I did a text analysis of the book A Chirstmas Carol by Charles Dickens and the text itself had lots of he said quotes and lots of apostrophes. There's always a work around. You can use double quotes to enclose a string with an apostrophe. Use single quotes to enclose a string with double quotes. There are also special escape codes.

First line names, example

```
name x y
Alpha 4 8
Bravo 8 16
Charlie 12 24
Delta 16 32
Echo 24 48
```

First line names, SAS code

```
filename raw_data
  "../data/first-line-names.txt";
libname module01
  "../data";
proc import
    datafile=raw_data dbms=dlm
    out=module01.first_line_names replace;
  delimiter=" ";
  getnames=yes;
run;

proc print
    data=module01.first_line_names(obs=2);
  title1 "First two rows of data";
run;
```

Here is the SAS code to read and print this file.

string to numeric dataset

```
name x y
Alpha 4 8
Bravo 8 NA
Charlie 12 24
Delta 16 32
Echo 24 48
```

import-string-to-numeric.sas (1 of 5)

```
ods pdf file="../src/import-string-to-
numeric.pdf";
filename raw_data
  "../data/string-to-numeric.txt";
libname module01 "../data";
```

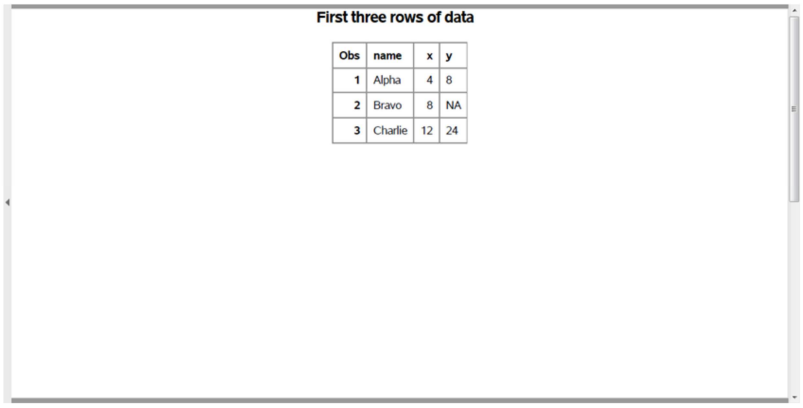
Here are the standard headers. Tell SAS where to store the output, where to find the data set and where to place any data files it creates.

import-string-to-numeric.sas (2 of 5)

```
proc import
    datafile=raw_data dbms=dlm
    out=module01.string_to_numeric replace;
  delimiter=" ";
  getnames=yes;
run;

proc print
    data=module01.string_to_numeric(obs=3);
  title1 "First three rows of data";
run;
```

You might be tempted to read in the data using proc import. Normally that's what you'd prefer for reading in data when the variable names are the first line of the data file. But when you do this you lose control over designating which variables are numeric and which are strings. SAS will make an educated guess, but the NA code for missing in the second observation will trip SAS up.

import-string-to-numeric.sas (3 of 5)



First three rows of data

Obs	name	x	y
1	Alpha	4	8
2	Bravo	8	NA
3	Charlie	12	24

Output from proc print

The NA code confused proc import. It saw a column that had some numbers in it and some letters, so it thought “string”. If SAS were smart enough to recognize the NA code as missing, then it would have been okay, but NA is a code used by R, and it would be unfair for SAS to know the missing value codes for R and every other statistical package out there.

You can tell that the variable Z is a string because it is left justified. Notice that the 8 and the N and the 2 all line up.

import-string-to-numeric.sas (4 of 5)

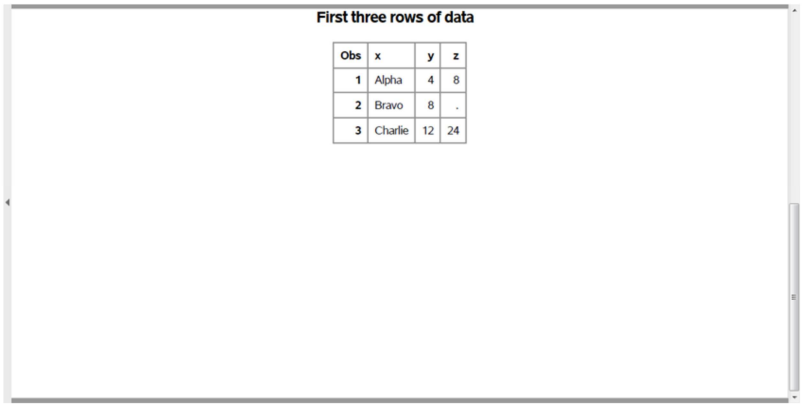
```
data module01.string_to_numeric;
  infile raw_data delimiter=" " firstobs=2;
  input
    x $
    y
    z ??;
run;

proc print
    data=module01.string_to_numeric(obs=3);
  title1 "First three rows of data";
run;
ods pdf close;
```

Instead use the data step because on the input statement, you can designate x as string, y and z as numeric. Since some of the values for z may not look numeric, warn SAS so that SAS doesn't go bonkers with flags and warnings.

You do this with a double question mark.

Also, since the data starts with variable names, you have to tell SAS to skip the first line and start reading at the second line. This is what the firstobs=2 option does.

import-string-to-numeric.sas (5 of 5)



First three rows of data

Obs	x	y	z
1	Alpha	4	8
2	Bravo	8	.
3	Charlie	12	24

Output from proc print

With this code, the variable z is input as numeric. Notice the right justification. The 8 and the . and the 4 all line up.) The NA code is converted to the SAS missing value code. Everyone is happy.

There are other ways you can work around this problem, but this is simple and clean.

Break #1

- What have you learned.
  - Tab delimited files
  - Fixed width files
  - First line variable names
  - Converting strings to numeric
- What's coming next
  - Two real world examples
    - Abalone age prediction
    - Saratoga house prices

Read in the abalone data file

- Abalone age prediction
  - This is a comma separated value file without variables names at the top. You can find the variable names in a data dictionary published at second link.
    - https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data
    - https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.names

abalone data, first ten lines

```
M,0.455,0.365,0.095,0.514,0.2245,0.101,0.15,15
M,0.35,0.265,0.09,0.2255,0.0995,0.0485,0.07,7
F,0.53,0.42,0.135,0.677,0.2565,0.1415,0.21,9
M,0.44,0.365,0.125,0.516,0.2155,0.114,0.155,10
I,0.33,0.255,0.08,0.205,0.0895,0.0395,0.055,7
I,0.425,0.3,0.095,0.3515,0.141,0.0775,0.12,8
F,0.53,0.415,0.15,0.7775,0.237,0.1415,0.33,20
F,0.545,0.425,0.125,0.768,0.294,0.1495,0.26,16
M,0.475,0.37,0.125,0.5095,0.2165,0.1125,0.165,9
F,0.55,0.44,0.15,0.8945,0.3145,0.151,0.32,19
```

import-abalone.sas (1 of 3)

```
ods pdf file="../results/import-abalone.pdf";
filename raw_data
  "https://archive.ics.uci.edu/ml/machine-
learning-databases/abalone/abalone.data";
libname module01
  "../data";
```

Import the Abalone age prediction file.

Tell SAS where to find the data. Notice that the filename statement includes a directl link to the website that has this data.

import-abalone.sas (2 of 3)

```
data module01.abalone;
  infile raw_data delimiter=",";
  input
    Sex $
    Length
    Diameter
    Height
    Whole_weight
    Shucked_weight
    Viscera_weight
    Shell_weight
    Rings;
run;
```

Use the data step because the names of the variables are not on the first line of the file.

This file uses commas for delimiters. Specify this on the infile statement.

There are nine variables in this data set. Sex is a string, a single character with values F for female, I for infant, and M for male. The other variables are numeric.

import-abalone.sas (3 of 3)

```
proc print
    data=module01.abalone(obs=10);
  var Sex Length Diameter Height Whole_weight;
  title1 "First ten rows of data";
run;
ods pdf close;
```

If the import is successful, you will see a small part of the data: the first ten rows and the first five variables.

Read in the Saratoga house prices file

- Saratoga house prices
  - This is a tab delimited file with the names of the variables in the first row. The second link provides a brief description of the data.
    - https://dasl.datadescription.com/download/data/3437
    - https://dasl.datadescription.com/datafile/saratoga-house-prices/

saratoga data, first ten lines

Price	Size	Baths	Bedrooms		Fireplace	Acres	Age
142.21200	1.9820000	1	3	0	2	133	
134.86500	1.6760000	1.5	3	1	0.38		14
118.00700	1.6940000	2	3	1	0.96		15
138.29700	1.8000000	1	2	1	0.48		49
129.47000	2.0880000	1	3	1	1.84		29
206.51200	1.4560000	2	3	0	0.98		10
50.709000	0.96000000	1.5	2	0	0.01		12
108.79400	1.4640000	1	2	0	0.11		87
68.353000	1.2160000	1	2	0	0.61		101

import-saratoga.sas (1 of 3)

```
ods pdf file="../results/import-saratoga.pdf";
filename raw_data

"https://dasl.datadescription.com/download/data/3
437";
libname module01
  "../data";
```

This is the first few lines of code, showing where to store the output, where to find the input and where to store the SAS binary data set the program creates.

Notice that the filename statement points directly to the website.

import-saratoga.sas (2 of 3)

```
proc import
    datafile=raw_data dbms=dlm
    out=module01.saratoga replace;
  delimiter="09"x;
  getnames=yes;
run;
```

This is a tab delimited file with variable names on the first row. So use proc import with the getnames=yes. The delimiter for tab is denoted by “09”x.

import-saratoga.sas (3 of 3)

```
proc print
    data=module01.saratoga(obs=10);
  title1 "First two rows of data";
run;
ods pdf close;
```

If the import is successful, you will see a small part of the data: the first ten rows and the first five variables.

Summary

– What have you learned
  - Reading delimited files
  - Reading fixed width files
  - First line variable names
  - Converting strings to numeric