simon-5507-03-slides

Topics to be covered

- What you will learn
  - Importing space-delimited files
  - Importing comma-delimited files
  - Importing tab-delimited files
  - Importing files with other delimiters
  - Importing files with strings
  - Fixed width files
  - Files with names in first row
  - Troubleshooting

Defining a macro variable

- %let command
  - defines a macro value
- &macro_variable_name
  - substitutes that macro value in the code

Speaker notes

I find myself switching often between SAS On Demand for Academics and SAS on Remote Labs. One thing to simplify the code is to create a macro variable. A macro variable is a value that you specify once, typically right at thestart of your program code, and then use repeatedly later in your code.

Simple example

```
%let row_limit=10;

(later on in your code)

proc print
    data=phase1(obs=&row_limit);

(this gets converted by SAS into)

proc print
    data=phase1(obs=10);
```

Speaker notes

Here’s a simple example. You set a macro variable called row_limit equal to the string 10. Then later in your program, you include that macro variable as part of proc print to specify that

When should you use a macro variable?

- Document important options at the top of your program
  - Quick and easy changes
- Reduce the amount of repetitive typing
- Macro variables automatically available
  - My favorites, &sasdate and &sasver

Speaker notes

Macro variables have several uses. First, they allow you to place important options near the top of your code rather than buried somewhere in the code depths. Because they are easy to find, they allow you to quick changes. Finally, sometimes a piece of SAS code gets used multiple times. A macro variable allows you to write it once and then use the shorter macro name over and over again. This can sometimes save a lot of repetitive typing.

SAS has a wide range of macro variables. Most of these are only needed for very advanced and very specialized applications. You've already seen two of these,

Text files

- Each row is a separate observation
- How to specify individual columns?
  - Blanks
  - Delimiters
  - Fixed width

Speaker notes

There’s a quip, “Time is nature’s way of keeping everything from happening all at once.” For text file that represents data, how do you keep one piece of data separate from another piece of data?

The first thing that is commonly done is to put each patient’s data on a separate line. You don’t have to do it this way, but it works quite well for many types of data.

Within a line, how do you keep one piece of data for a patient separate from another? The simplest choice, and one that works well with SAS is to separate using one or more blanks.

You can use other marks, called delimiters, to separate pieces of data. The most common delimiter is a comma.

You can also specify which pieces of data go into which columns. This is fixed width format.

Importing choices

- Add options to the infile statement
- Use proc import
  - Especially useful when the first row includes variable names
  - Also useful for binary data files (e.g., Excel)
- Use proc sql (not covered in this class)
- Manual reformatting
  - Global search and replace
  - Not usually a good idea

Speaker notes

Often, you will just need to add an option to the infile statement to get SAS to read a particular text file.

You might also need to use the import procedure. This procedure is especially helpful when your first line of data includes the variable names.

While we will not cover it in this class, the import procedure also lets you read in binary data files. A binary file is a file that uses a special storage approach that has some advantages. It often takes up less space and can be read in a lot faster. But the hallmark of a binary file is that you can’t view it in a text editor like Notepad or TextEdit.

Sometimes a text file is difficult to import. 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.

Options during data import

- Skipping rows
- Converting strings to numbers

Speaker notes

Sometimes you have to skip a few rows. Sometimes you have to convert strings to numbers. There are options for doing this in SAS.

Space delimited, example

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

Speaker notes

Here is an example of a space-delimited file. It uses one or more blanks to separate the indidvidual data values.

Documentation header (1)

```
* 5507-03-simon-demo-01-space-delimited.sas
* author: Steve Simon
* creation date: 2019-07-01
* purpose: to import data with spaces as delimiters
* license: public domain;
```

Speaker notes

- Comments on the code: Documentation header

This is my standard for the documentation header. It specifies the filename, author, creation date, purpose, and license.;

File locations (2)

```
%let path=q:/5507-2025b/03;

ods pdf file=
    "&path/results/5507-03-simon-demo-01-space-delimited.pdf";

libname perm
    "&path/data";

filename raw_data
    "&path/data/text-space-delimited.txt";
```

Speaker notes

- Comments on the code: File locations

Since all three file locations (ods, libname, and filename) use the same path, a macro variable can simplify things. The %let command creates the macro variable path, and anywhere that SAS sees an &path, it replaces it with the proper file location.;

Read space-delimited file (3)

```
data perm.space_delimited;
  infile raw_data;
  input x y z;
run;

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

ods pdf close;
```

Speaker notes

- Comments on the code: Read space-delimited file

Files with one or more blanks between each data value are easy to read in SAS and require no special code;

Live demo, 1

Break #1

- What you have learned
  - Importing space-delimited files
- What’s coming next
  - Importing comma-delimited files

A comma delimited file

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

Speaker notes

This is the same artificial dataset, using commas as delimiters. This is the most common type of file that you will find on the Internet.

Documentation header (1)

```
* 5507-03-simon-demo-02-comma-delimited.sas
* author: Steve Simon
* creation date: 2019-07-01
* purpose: to import comma delimited files
* license: public domain;
```

Speaker notes

- Comments on the code: Documentation header

This is my standard for the documentation header. It specifies the filename, author, creation date, purpose, and license.;

File locations (2)

```
%let path=q:/5507-2025b/03;

ods pdf file=
    "&path/results/5507-03-simon-demo-02-comma-delimited.pdf";

libname perm
    "&path/data";

filename raw_data
    "&path/data/text-comma-delimited.csv";
```

Speaker notes

- Comments on the code: File locations

This code shows where to print output, where to store data, and where to find data.;

Specifying a comma delimiter (3)

```
data perm.comma_delimited;
  infile raw_data delimiter=",";
  input x y z;
run;

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

ods pdf close;
```

Speaker notes

- Comments on the code: Specifying a comma delimiter

Adding the delimiter keyword to the infile statement informs SAS that individual data values in a row of text are separated or delimited by a specific character. In this code, the delimiter is a comma.;

Live demo, 2

- delimiter argument (infile subcommand, data step)

Break #2

- What you have learned
  - Importing comma-delimited files
- What’s coming next
  - Importing tab-delimited files

Tabs are evil

Speaker notes

Tabs were used on manual typewriters as a time saving device. If you wanted to indent the first line of a paragraph, you could just type five spaces (maybe ten) and then start typing. The tab key on a manual typerwriter would jump automatically to the indent location.

When computer keyoboards came along, the powers-that-be decided to include the tab key. It seemed like a good idea. So good that they designated a special character in ASCII to represent it’s heritage.

The binary representation

- Tab character: Binary 0000 1001 or hexadecimal 09
- Space character: Binary 0010 0000 or hexadecimal 20
- Tabs and spaces look similar, but
  - A tab delimited file is not a space delimited file
  - A space delimited file is not a tab delimited file.

Speaker notes

The tab character in ASCII is represented by binary 0000 1001 or hexadecimal 09, while the space key is represented by binary 0010 0000 or hexadecimal 20.

Designations for the tab character

- “09”X
- ^t
- \t
- 0x09

Speaker notes

If you want to work with the tab character, you need to know the code that your software system uses for the tab and other special characters.

In SAS, the code is “09”X. In Microsoft Word, it is ^t. In R it is either r 0x09. The latter also works with Python, C, and other major programming languages.

Epilog: The non-breaking space

- Looks just like a regular space
  - Does not allow word wrap
  - Does not allow multiple blanks to collapse
- Binary 1010 0000, Hexadecimal A0
- Represented in html as ` `

Speaker notes

You may encounter another problematic character, the non-breaking space. These look like the space character and the only difference is that you can’t use word wrap with a non-breaking space. This forces the word to the left and the word to the right of the non-breaking space to stay on the same line.

Also, a non-breaking space is preserved in some systems like html which would otherwise collapse multiple blanks into a single blank.

The non-breaking space will sometimes hide among the regular spaces, but since it is not a delimiter, it will often mess up your import efforts.

You won’t encounter a non-breaking space too often, but when you do, it is very hard to spot and to fix.

A tab delimited file

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

Speaker notes

This is the same artificial dataset, using tabs as delimiters. This is also a very common type of file that you will find on the Internet.

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.

Documentation header (1)

```
* 5507-03-simon-demo-03-tab-delimited.sas
* author: Steve Simon
* creation date: 2019-07-01
* purpose: to import a tab-delimited file into SAS
* license: public domain;
```

Speaker notes

- Comments on the code: Documentation header

This is my standard for the documentation header. It specifies the filename, author, creation date, purpose, and license.;

File locations (2)

```
%let path=q:/5507-2025b/03;

ods pdf file=
    "&path/results/5507-03-simon-demo-03-tab-delimited.pdf";

libname perm
    "&path/data";

filename raw_data
  "&path/data/text-tab-delimited.txt";
```

Speaker notes

- Comments on the code: File locations

This code shows where to print output, where to store data, and where to find data.;

Specifying a tab delimiter (3)

```
data perm.tab_delimited;
  infile raw_data delimiter="09"X;
  input x y z;
run;

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

ods pdf close;
```

Speaker notes

- Comments on the code: Specifying a tab delimiter

The code “09”X designates a tab as the delimiter.;

Live demo, 3

- “09”X (SAS code for the tab character)

Break #3

- What you have learned
  - Importing tab-delimited files
- What’s coming next
  - Importing files with other delimiters

A tilde delimited file

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

Speaker notes

This is the same artificial dataset, using the tilde symbol (~) as the delimiter. You might use this when a more common symbol, like a comma, would lead to confusion.

Documentation header (1)

```
* 5507-03-simon-demo-04-tilde-delimited.sas
* author: Steve Simon
* creation date: 2019-07-01
* purpose: to import a tilde-delimited file into SAS
* license: public domain;
```

Speaker notes

- Comments on the code: Documentation header

This is my standard for the documentation header. It specifies the filename, author, creation date, purpose, and license.;

File locations (2)

```
%let path=q:/5507-2025b/03;

ods pdf file=
    "&path/results/5507-03-simon-demo-04-tilde-delimited.pdf";

libname perm
  "&path/data";

filename raw_data
  "&path/data/text-tilde-delimited.txt";
```

Speaker notes

- Comments on the code: File locations

This code shows where to print output, where to store data, and where to find data.;

Specifying a tilde delimiter (3)

```
data perm.tilde_delimited;
  infile raw_data delimiter="~";
  input x y z;
run;

proc print
    data=perm.tilde_delimited(obs=2);
  title1 "First two rows of data";
run;

ods pdf close;
```

Speaker notes

- Comments on the code: Specifying a tilde delimiter

Anything, including the tilde symbol (~) can be a delimiter.;

Live demo, 4

- “09”X (SAS code for the tab character)

Break #4

- What you have learned
  - Importing files with other delimiters
- What’s coming next
  - Importing files with strings

String data

- A combination of letters, numbers, blanks, and/or symbols,
  - Examples
    - Simon, Stephen D.
    - 2411 Holmes Street
    - 816. 235-6617
- Also called character data
- Often enclosed in single quotes (') or double quotes (")
  - Avoid smart/curly quotes (‘ ’ or “ ”)

Speaker notes

String data, also called character data, is a combination of letters, numbers and/or symbols. A patients name, address, and telephone number, all of which are a combination of numbers, letters, and symbols, are examples of data that need strings to represent it.

Cautions about strings

- Strings longer than 8 characters
- Strings that include delimiters
- Strings that include quote marks (Conan O’Brian)
- Symbols outside 7-bit ASCII
  - resumé, naïve, façade
- Strings with different alphabets

Speaker notes

SAS (and SPSS) require a bit of care, at times, with strings longer than eight characters. Most of the time, longer strings are okay, but once in a while, you have to take special account for the longer strings.

Strings can (and often do) include common delimiters like the space and the comma.

Strings can also include single or double quote marks as part of the string. Common Irish names, like Conan O’Brian, are one example.

Also be careful with strings outside the 7-bit ASCII code. The term 7-bit ASCII refers to the original character set associated with many early computers. These were limited to only 128 because of computer limits and did not allow for accented characters (resumé), diaeresis (naïve), or the cedilla (façade). Several competing standards were developed for these special characters, and sometimes they clash.

Strings using different alphabets like Cyrillic, Greek, Hebrew, Arabic, Chinese will also sometimes present difficulties. These alphabets may sometimes require more than 8 bytes (especially Chinese) and the “size” of a string is sometimes tricky to measure.

A file with strings

```
Alpha,4,8
Bravo,8,16
Charlie,12,24
Delta,16,32
Echo,24,48
```

Speaker notes

This is the same artificial dataset, with no delimiters. There are a few stray blanks, but only because some of the numbers are one digit instead of two or two digits instead of three.

Documentation header (1)

```
* 5507-03-simon-demo-05-string-data.sas
* author: Steve Simon
* creation date: 2019-07-02
* purpose: to import data that includes a string
* license: public domain;
```

Speaker notes

- Comments on the code: Documentation header

This is my standard for the documentation header. It specifies the filename, author, creation date, purpose, and license.;

File locations (2)

```
%let path=q:/5507-2025b/03;

ods pdf file=
    "&path/results/5507-03-simon-demo-05-string-data.pdf";

libname perm
    "&path/data";

filename raw_data
    "&path/data/text-with-strings.txt";
```

Speaker notes

- Comments on the code: File locations

This code shows where to print output, where to store data, and where to find data.;

Reading string data (3)

```
data perm.string_data;
  infile raw_data delimiter=",";
  input
    name $
    x
    y;
run;

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

ods pdf close;
```

Speaker notes

- Comments on the code: Reading string data

Use the dollar sign ($) to designate a particular variable as being string rather than numeric.;

Live demo, 5

Break #5

- What you have learned
  - Importing files with strings
- What’s coming next
  - Fixed width files

A fixed-width file

```
 4 8 40
 816 80
1224120
1632160
2448240
```

Speaker notes

This is the same artificial dataset, using the tilde symbol (~) as the delimiter. You might use this when a more common symbol, like a comma, would lead to confusion.

Documentation header (1)

```
* 5507-03-simon-demo-06-fixed-width.sas
* author: Steve Simon
* creation date: 2019-07-01
* purpose: to import data in a fixed width format
* license: public domain;
```

Speaker notes

- Comments on the code: Documentation header

This is my standard for the documentation header. It specifies the filename, author, creation date, purpose, and license.;

File locations (2)

```
%let path=q:/5507-2025b/03;

ods pdf file=
    "&path/results/5507-03-simon-demo-06-fixed-width.pdf";

libname perm
    "&path/data";

filename raw_data
    "&path/data/text-fixed-width.txt";
```

Speaker notes

- Comments on the code: File locations

This code shows where to print output, where to store data, and where to find data.;

Reading fixed width data (3)

```
data perm.fixed_width;
  infile raw_data delimiter=",";
  input
    x 1-2
    y 3-4
    z 5-7;
run;

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

ods pdf close;
```

Speaker notes

- Comments on the code: Reading fixed width data

There are several options for reading fixed width data. This code shows how to use a start and end location for each variable. The variable X is located in columns 1 and 2. The variable y is located in columns 3 and 4. The variable z uses three columns: 5, 6, and 7.;

Live demo, 6

- “09”X (SAS code for the tab character)

Break #6

- What you have learned
  - Fixed width files
- What’s coming next
  - Files with names in first row

A file with names in the first line

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

Speaker notes

This is the same artificial dataset, with the variable names (x, y, and z) as the first line of the file.

Documentation header (1)

```
* 5507-03-simon-demo-07-first-line-names.sas
* author: Steve Simon
* creation date: 2019-07-02
* purpose: to import data with variable names on the first line
* license: public domain;
```

Speaker notes

- Comments on the code: Documentation header

This is my standard for the documentation header. It specifies the filename, author, creation date, purpose, and license.;

File locations (2)

```
%let path=q:/5507-2025b/03;

ods pdf file=
    "&path/results/5507-03-simon-demo-07-first-line-names.pdf";

libname perm
    "&path/data";

filename raw_data
    "&path/data/text-first-line-names.csv";
```

Speaker notes

- Comments on the code: File locations

This code shows where to print output, where to store data, and where to find data.;

The import procedure (3)

```
proc import
    datafile=raw_data
    dbms=dlm
    out=perm.first_line_names
    replace;
  delimiter=",";
  getnames=yes;
run;

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

Speaker notes

- Comments on the code: The import procedure

The import procedure works with both text files and binary files. The data=option tells SAS the location of the text or binary data

The dbms=delim option tells SAS that this is a delimited text file.

The replace option tells SAS that it is okay to write this data set on top of another dataset with the same name, if it exists.

The delimiter subcommand tells SAS to use a comma (,) as a delimiter.

The getnames=yes subcommand tells SAS to use the first line of the file as the variable names.;

Live demo, 7

- The import procedure
  - The data= option
  - The out= option
  - The replace option
- The dbms subcommand
- The getnames subcommand

Break #7

- What you have learned
  - Files with names in first row
- What’s coming next
  - Troubleshooting

Troubleshooting

- Mistakes often will produce error or warning messages
  - Always check the log window!
- Using the wrong delimiter
  - Data smashed together in a single variable
- Misspecifying first line variable names
  - Strings as first line of data
  - Variable names are actually numbers
- Nonstandard missing value codes
  - Something other than dot (.)
  - Numeric data converted to strings

Speaker notes

If you make a mistake in your code, SAS will usually catch it. Once in a while, though, it tries to make the error fit the data, with sometimes cryptic results.

Now I want to encourage you to get in the habit of looking at the log window before looking at the output. If you get any error or warning messages, fix things. Don’t turn in output that may be compromised.

If you specify the wrong delimiter, there will be only one variable per line. This often triggers an error. But sometimes, you might see all the individual data smashed together in an ugly looking string.

If you tell SAS that the first line contains variable names (getnames=yes) and it doesn’t or if the first line does contrain variable names and you forget to tell SAS, then you might see some odd data sets. Your variable names could become the first line of actual data. Or the names of the variables might actually be numbers.

If you have a non-standard missing value, such as NA (which is what R uses), then SAS may treat all the values for that variable as strings.

Summary

- What you have learned
  - Importing space-delimited files
  - Importing comma-delimited files
  - Importing tab-delimited files
  - Importing files with other delimiters
  - Importing files with strings
  - Fixed width files
  - Files with names in first row
  - Troubleshooting