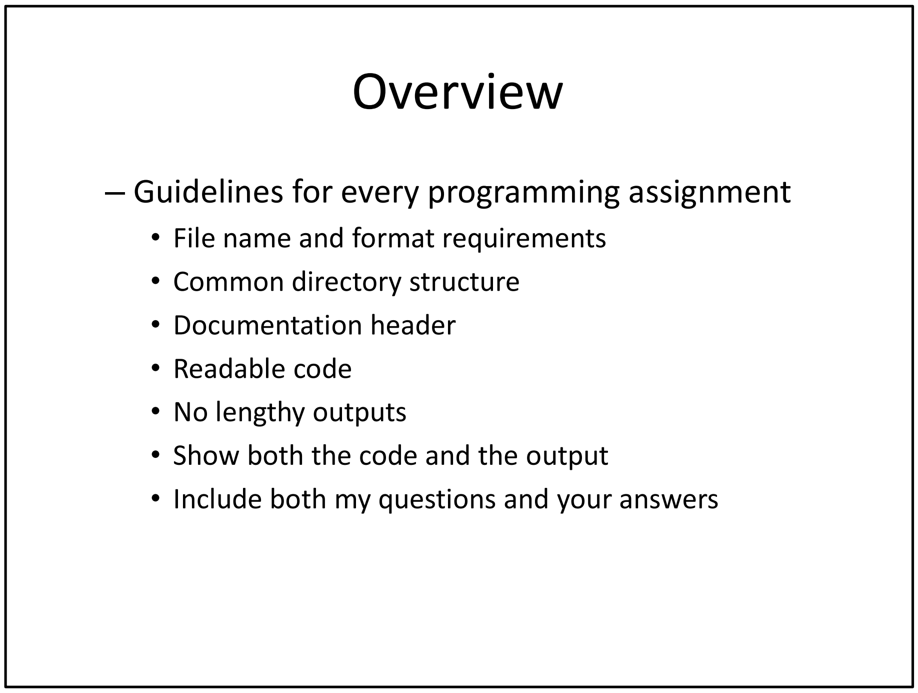
Basic programming expectations

Steve Simon

Created 2021-05-19



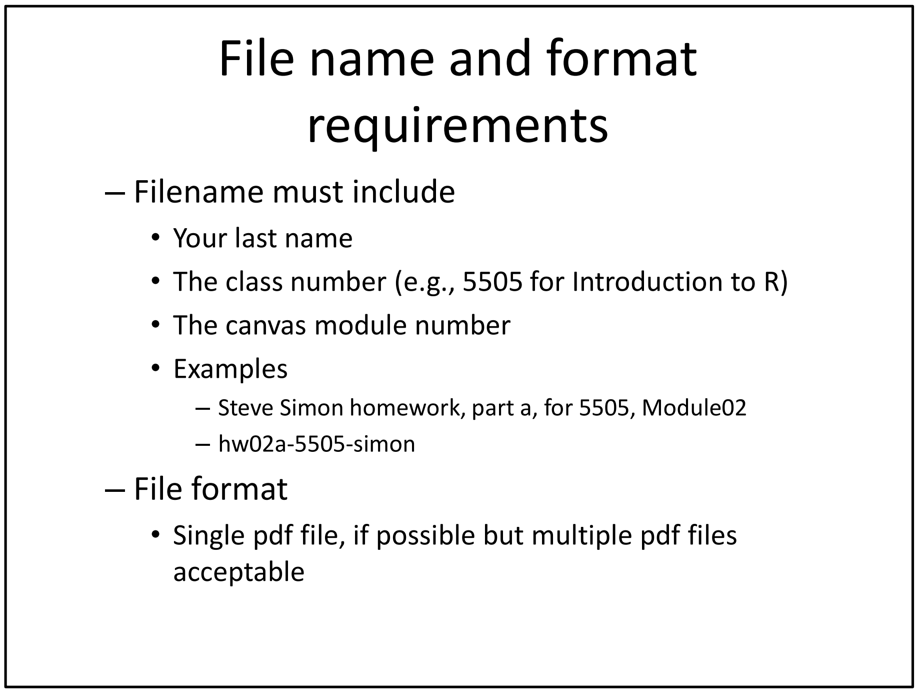
Overview

- Guidelines for every programming assignment
  - File name and format requirements
  - Common directory structure
  - Documentation header
  - Readable code
  - No lengthy outputs
  - Show both the code and the output
  - Include both my questions and your answers

If you learned programming like I did, you learned it with a bunch of bad habits that are very hard to break. I don't want you to develop these same bad habits, so I want you to adopt some good habits early.

Some of these items are not really important for developing good programming habits, but rather are for my convenience. Grading is a lot of work, and it is complicated by having some students attending two different classes of mine at the same time. There are a few simple things you can do that can help make my difficult job a bit easier.

There are seven different items that I want to cover in this talk.



File name and format requirements

- Filename must include
  - Your last name
  - The class number (e.g., 5505 for Introduction to R)
  - The canvas module number
  - Examples
    - Steve Simon homework, part a, for 5505, Module02
    - hw02a-5505-simon
- File format
  - Single pdf file, if possible but multiple pdf files acceptable

I teach multiple classes and sometimes the same student will be in two of my classes simultaneously. To avoid confusion, please make sure that you include certain information in any file that you submit to me on Canvas.

You need at least your last name, and if your last name is fairly common, either here or in your home country, your first name as well.

You need to indicate the four digit number for the class. The classes that I currently teach are

MEDB 5505, for Introduction to R

MEDB 5507, for Introduction to SAS

MEDB 5508, for Introduction to SQL (co-taught with Suman Sahil)

MEDB 5510, for Clinical Research Methodology, though this class will not have any programming exercises.

All of my classes are divided into modules and you need to indicate which module your assignment came from.

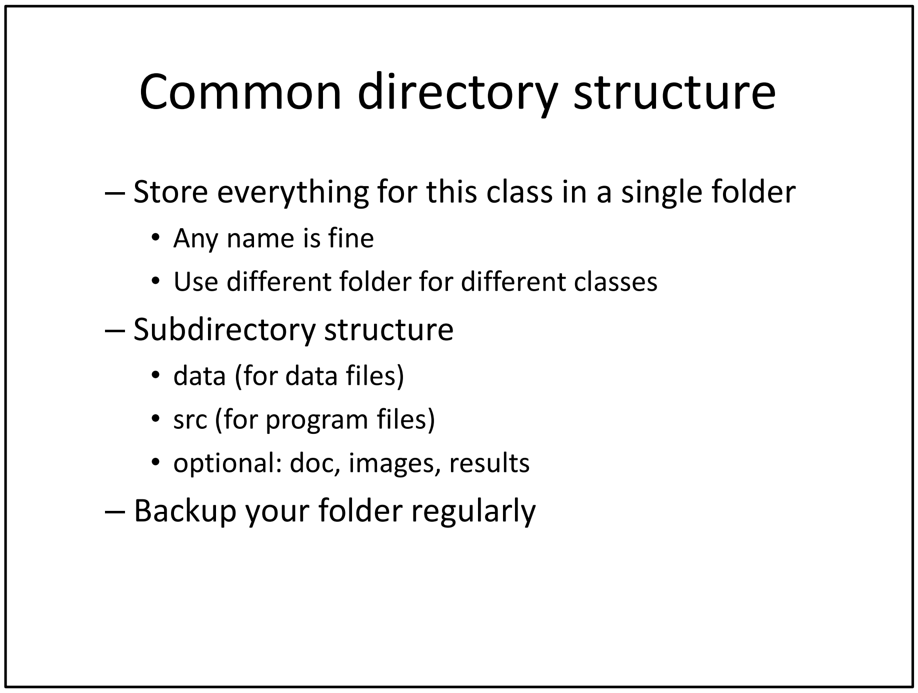
There are several ways you can name your files and meet these requirements. Here are two examples.

It is hard for me to grade if I have to juggle several different programs because student submit their assignments in several different formats. It really helps if you can produce pdf files, no matter what the original format of your work.

Ideally, you should combine all the pdf files into a single file.

There are a million different ways to create pdf files, and to combine multiple pdf files into a single file. I won’t go over them here, but if you have problems, let me know.

Now, if you are struggling to get an assignment in, don’t let the format requirements hold you back. I’d rather have an assignment turned in on time in any format rather than late in the required format.



Common directory structure

- Store everything for this class in a single folder
  - Any name is fine
  - Use different folder for different classes
- Subdirectory structure
  - data (for data files)
  - src (for program files)
  - optional: doc, images, results
- Backup your folder regularly

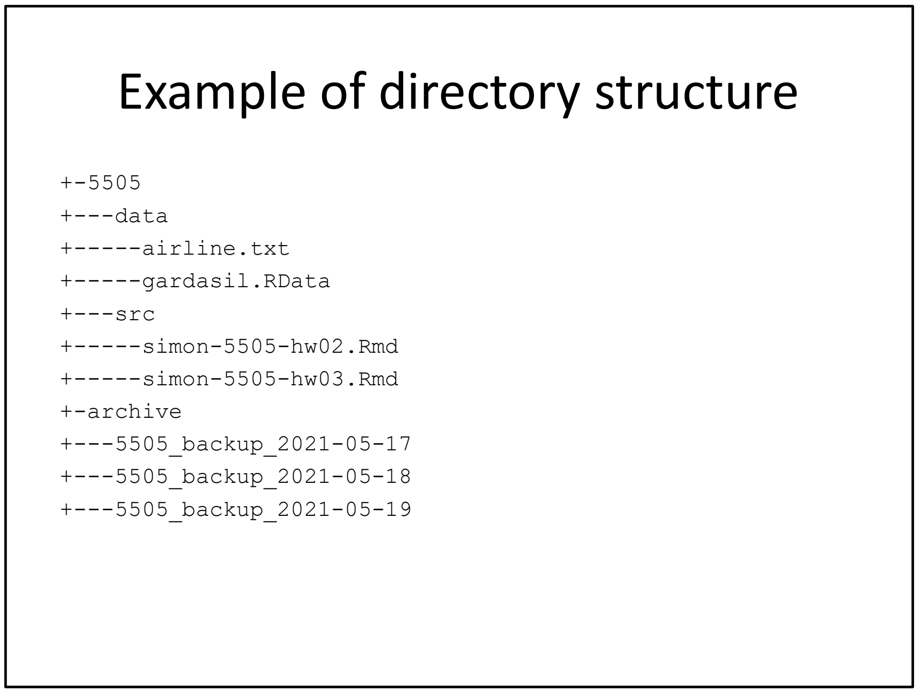
You should store all your programs, data, and output in a single folder with a specific subdirectory structure. Choose any name you want for your folder, but don't make two or more folders, unless this is done purposefully (such as a separate folder for each module).

For some classes, you may need to store this folder on a network drive.

If you are taking two different programming classes, please use two separate folders so that your programs don't clash.

It is very important to set up a common subdirectory structure. This will make things a lot easier for me when I try to answer questions about why your program doesn't work. The two key subdirectory folders that you must use are data (for data files) and src (for program files). You are welcome to set up additional folders, such as images for graphs stored as image files and doc for important documentation details.

Backup your folder regularly. The simplest way to do this is to have an archive folder and you just create subdirectories in the archive folder that are complete copies of the whole class folder.



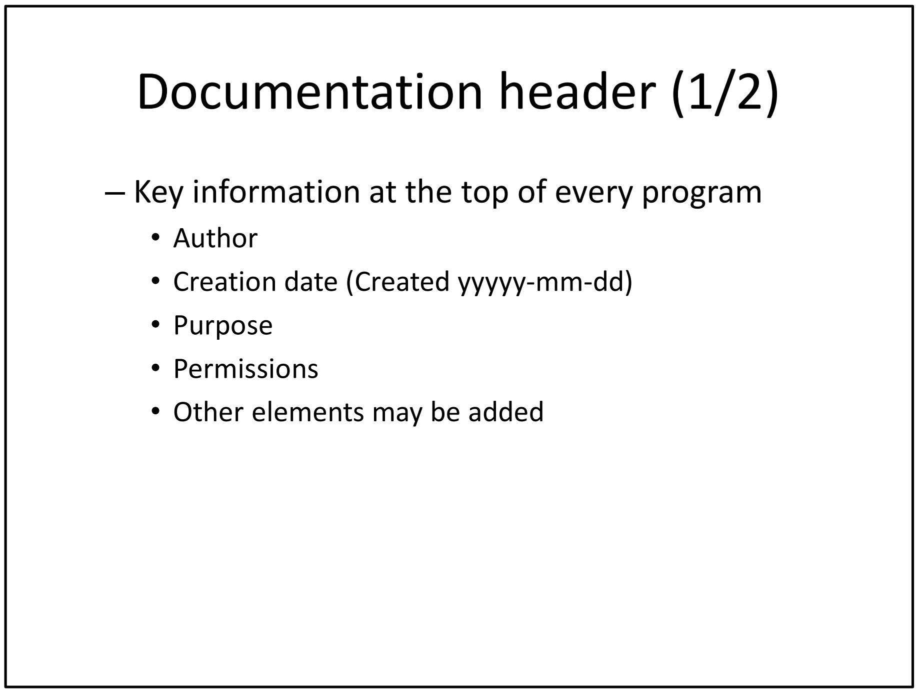
Example of directory structure

```
+-5505
+---data
+-----airline.txt
+-----gardasil.RData
+---src
+-----simon-5505-hw02.Rmd
+-----simon-5505-hw03.Rmd
+-archive
+---5505_backup_2021-05-17
+---5505_backup_2021-05-18
+---5505_backup_2021-05-19
```

Here’s an example of this directory structure. The main folder is names 5505.

The data subdirectory has two files: airline.txt, and gardasil.RData. There are two programs in the src subdirectory: simon-55505-hw02.Rmd and simon-5505-hw03.Rmd.

The archive folder has backups of the complete 5505 folder from May 17, 18, and 19 of 2021.



Documentation header (1/2)

– Key information at the top of every program
  - Author
  - Creation date (Created yyyyy-mm-dd)
  - Purpose
  - Permissions
  - Other elements may be added

At the top of every program, you should put information about the program that will help others understand important details.

The information that belongs in a documentation header is variable, but here are some of the things that I recommend.

Put your name in the documentation header. I will know it is from you, of course, because of how it is delivered to me via Canvas. But it is still a good habit to get into. If you are adapting someone else's code, you might offer them some credit here. In some settings you might want to include contact information (e.g., email address) as well.

Get in the habit of putting in the creation date. It is a bit silly perhaps in a program that you submit in a time limited setting like this class. It does become important when you are working on a project that spans multiple months or years.

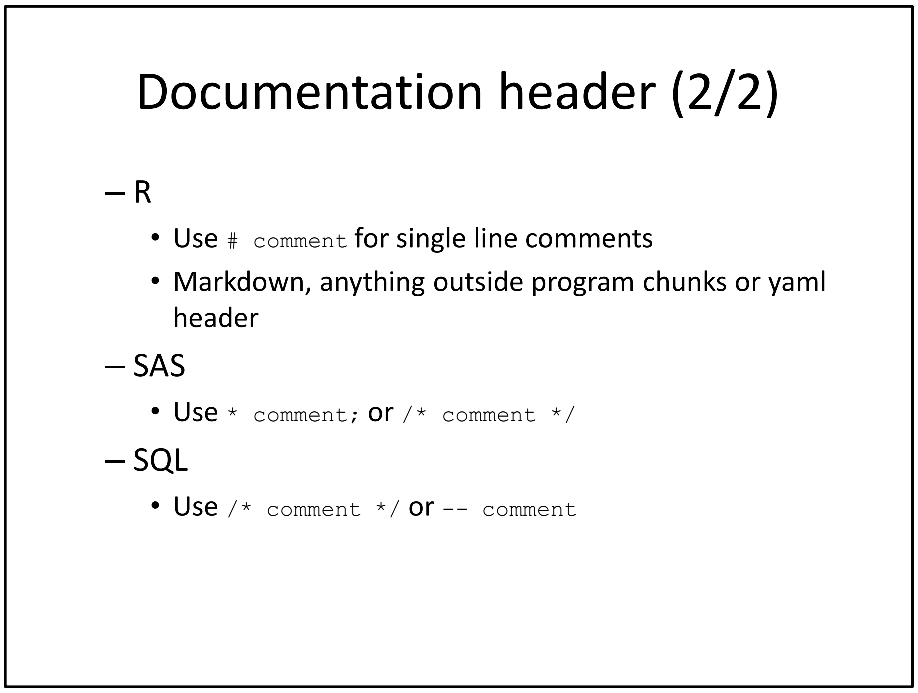
I'd encourage the use of the word "Created" so that any reader does not confuse this with the data last modified. Date last modified is sometimes a nice addition, especially on complex projects, but if you can't get the computer to insert a last modified date, then you may find this information a bit tedious to maintain.

Also, there is a standard format for dates, a four-digit year, a two-digit month, and a two-digit day with dashes (-) as separators. This is known as the ISO 8601 format, and I may or may not talk about this later in the class.

Describe the purpose of this program. Use something more descriptive than “Homework assignment,” such as “Read data from [some data source] and calculate an overall mean and standard deviation.”

It’s always provide permissions: what others can (or cannot) do with this program. For my class, I’d appreciate it if you let me know whether I can or cannot use anonymized pieces of your program as teaching examples in future classes. For programs in the real world, I would encourage you to place your programming code in the public domain, unless your employer has a different requirement. There are a a variety of open source licenses that are also good choices. I may or may not talk about licensing options later in this class.

Other elements of documentation may be added. It depends a lot on the context in which your program is being developed. I may or may not cover some of these additional elements later in the class.



Documentation header (2/2)

- R
  - Use `# comment` for single line comments
  - Markdown, anything outside program chunks or yaml header
- SAS
  - Use `* comment;` or `/* comment */`
- SQL
  - Use `/* comment */` or `-- comment`

Each program has a slightly differet way of adding comments.

In your R code, use the poind sign (#). Anything following this through to the end of the line will be considered a comment. If your comments extend across several lines, place the pound sign in front of each line of comments.

For R, however, I recommend that you use R Markdown. In R markdown, the R programming statements are placed inside program chunks. Anything outside a program chunk is considered a comment.

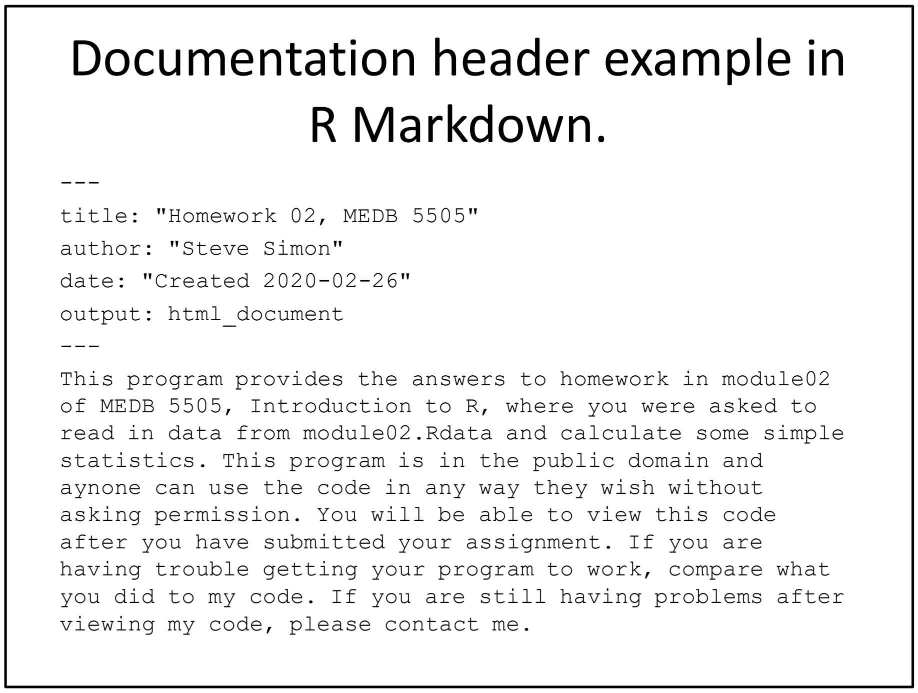
The header at the top of an R Markdown file, the yaml header, is also a place where you can include comments. Some of these comments, like the program author and the date will automatically appear in your output. Others will not, so be careful with the yaml header.

In SAS, you can use an asterisk (*) to start a comment, which only ends when SAS sees a semicolon. You can also start a SAS comment with a slash-asterisk and end it with an asterisk-slash. SAS comments automatically extend to a fresh line.

SQL also uses the slash-asterisk at the beginning and asterisk-slash at the end of a comment. You can also start a comment with a double dash (–) and this comment will extend to the end of a line.

If you are using R or SAS to access SQL, you can place your documentation header at the top of your R or SAS code.

You have a lot of flexibility with your documentation header, but if it is not included, or if it does not include the four items described above, you will lose points on your assignment.

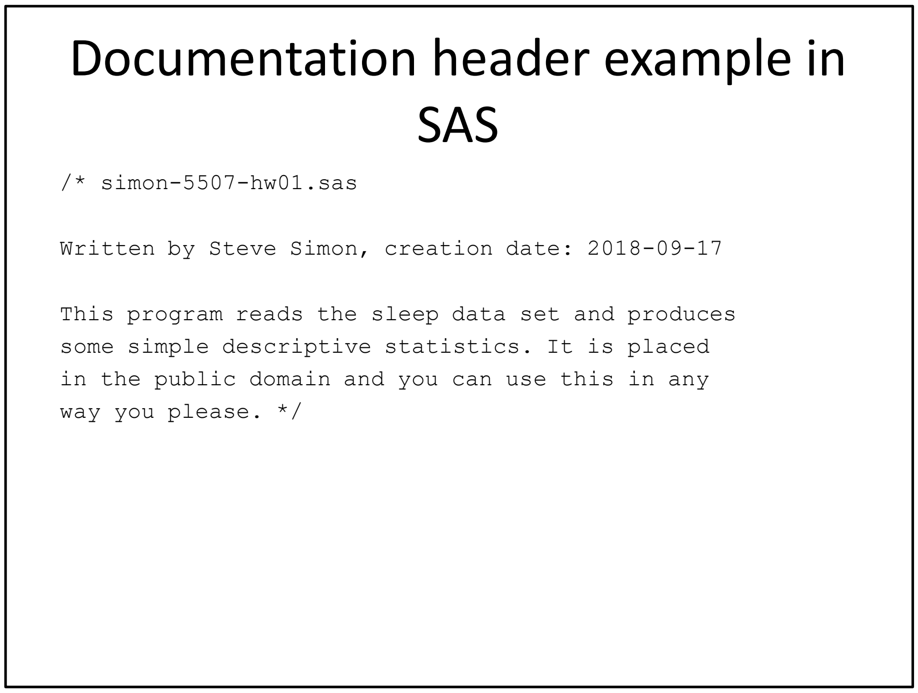


Documentation header example in R Markdown.

```
---
title: "Homework 02, MEDB 5505"
author: "Steve Simon"
date: "Created 2020-02-26"
output: html_document
---
This program provides the answers to homework in module02
of MEDB 5505, Introduction to R, where you were asked to
read in data from module02.Rdata and calculate some simple
statistics. This program is in the public domain and
aynone can use the code in any way they wish without
asking permission. You will be able to view this code
after you have submitted your assignment. If you are
having trouble getting your program to work, compare what
you did to my code. If you are still having problems after
viewing my code, please contact me.
```

Here’s an example of a documentation header using R Markdown. The author and creation date are in the yaml header. The first sentence below the yaml header describes the purpose of the program.

The second sentence places the code in the public domain so you can use the code any way that you might please.



Documentation header example in SAS

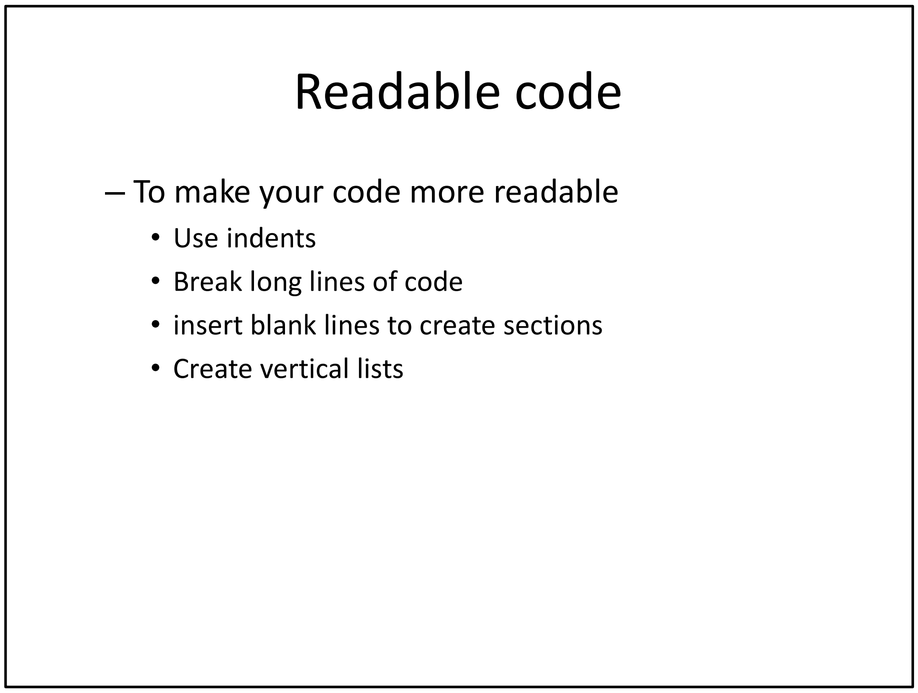
```
/* simon-5507-hw01.sas

Written by Steve Simon, creation date: 2018-09-17

This program reads the sleep data set and produces
some simple descriptive statistics. It is placed
in the public domain and you can use this in any
way you please. */
```

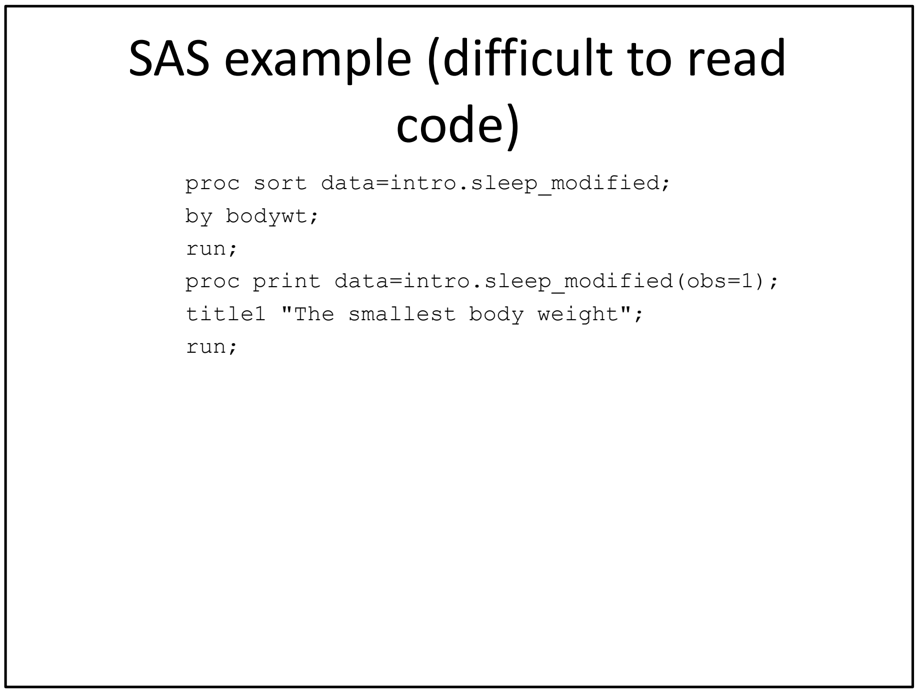
Here's an example using SAS. The comment starts with an asterisk and ends with a semi-colon. It covers the same details.

I won't show the code in SQL, but it would look similar



Readable code

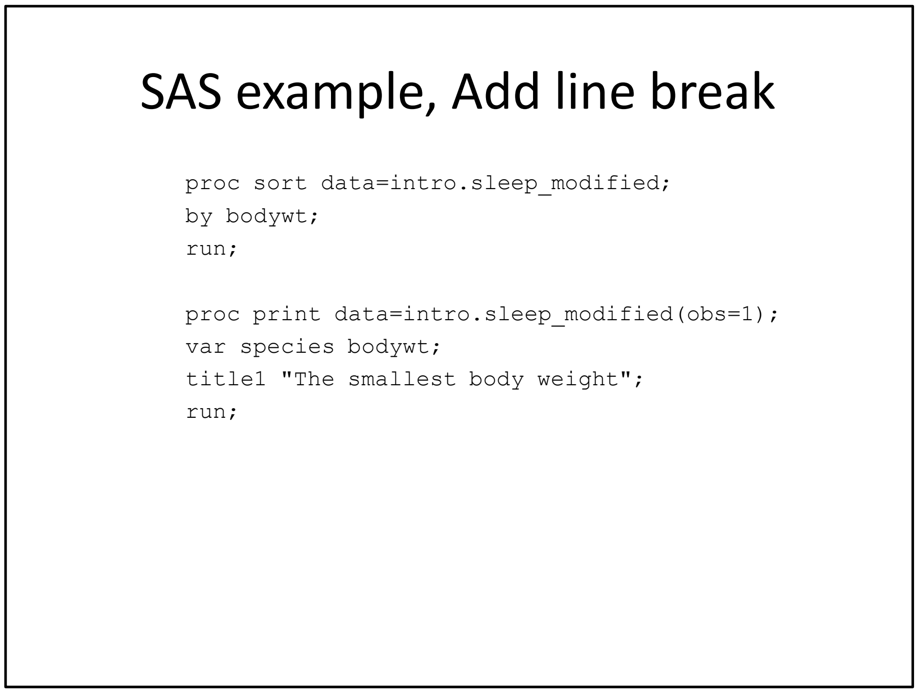
- To make your code more readable
  - Use indents
  - Break long lines of code
  - insert blank lines to create sections
  - Create vertical lists



SAS example (difficult to read code)

```
proc sort data=intro.sleep_modified;
by bodywt;
run;
proc print data=intro.sleep_modified(obs=1);
title1 "The smallest body weight";
run;
```

Here’s an example using the print procedure in SAS. It’s not too hard to read, but you can improve it quite a bit by adding line breaks, indenting, andshould

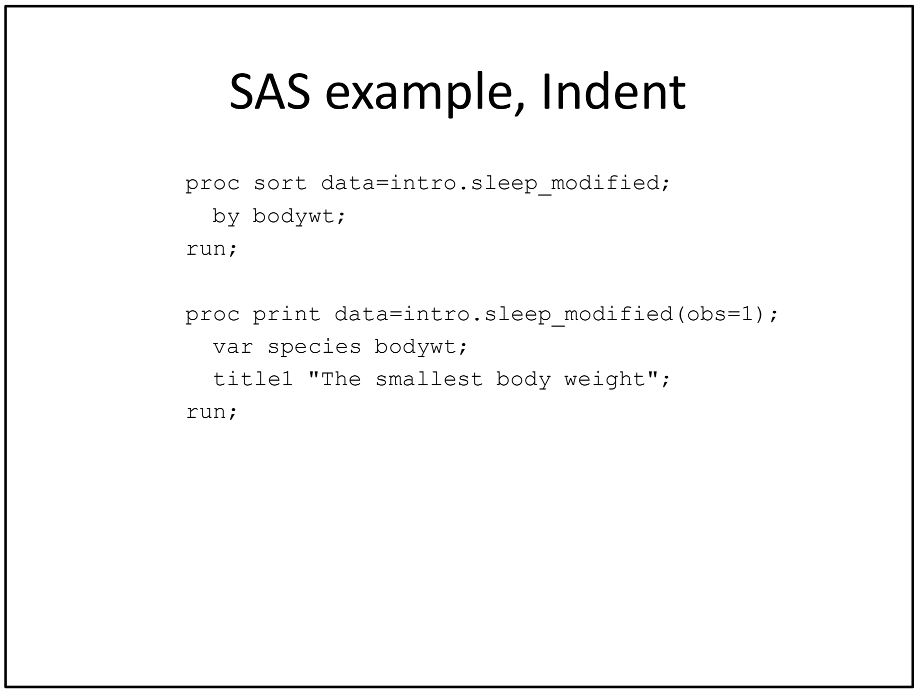


SAS example, Add line break

```
proc sort data=intro.sleep_modified;
by bodywt;
run;

proc print data=intro.sleep_modified(obs=1);
var species bodywt;
title1 "The smallest body weight";
run;
```

A blank link emphasizes the two steps in this program, sorting and printing.

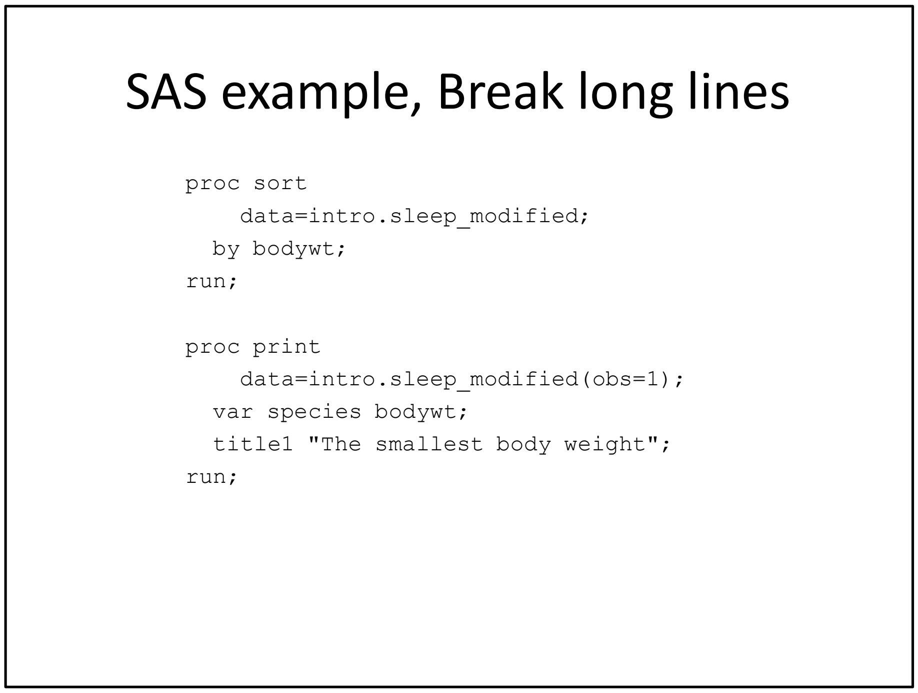


SAS example, Indent

```
proc sort data=intro.sleep_modified;
  by bodywt;
run;

proc print data=intro.sleep_modified(obs=1);
  var species bodywt;
  title1 "The smallest body weight";
run;
```

By indenting the by subcommand within the sort and the var and title1 subcommands in the print procedure, you further emphasize the desicrete steps in the program.



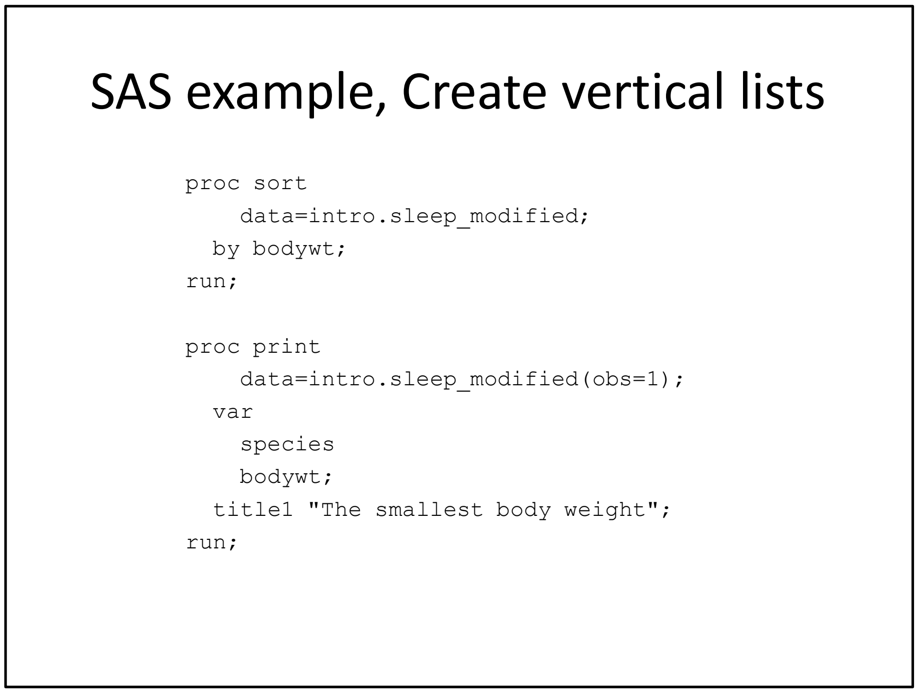
SAS example, Break long lines

```
proc sort
    data=intro.sleep_modified;
  by bodywt;
run;

proc print
    data=intro.sleep_modified(obs=1);
  var species bodywt;
  title1 "The smallest body weight";
run;
```

There is no line that is terribly long in this example, but you could split the proc sort and the proc print at data to make the line shorter.

I used a double indent here to emphasize that data is part of the previous procedure.

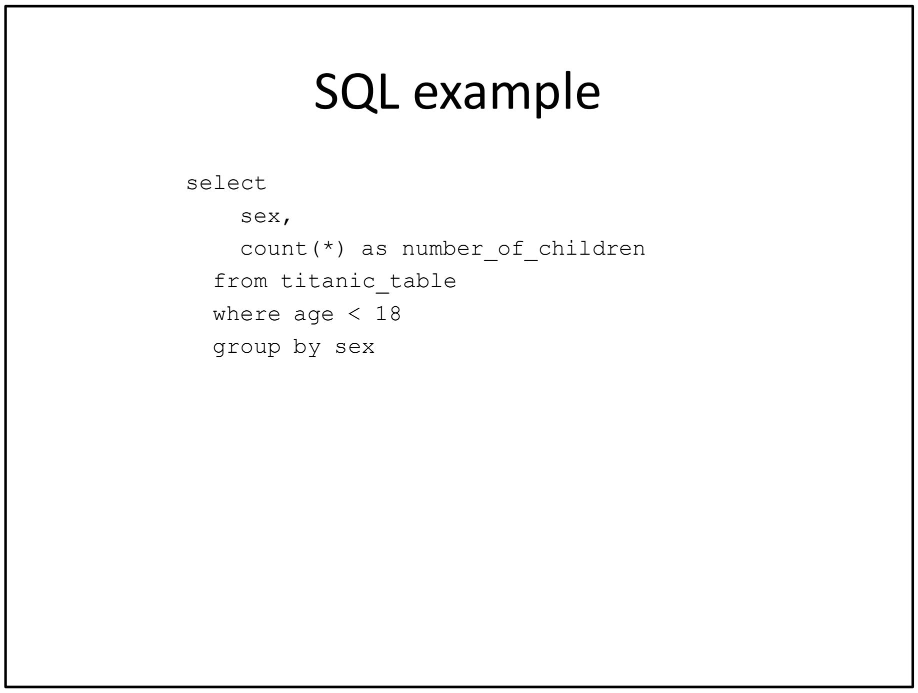


SAS example, Create vertical lists

```
proc sort
    data=intro.sleep_modified;
  by bodywt;
run;

proc print
    data=intro.sleep_modified(obs=1);
  var
    species
    bodywt;
  title1 "The smallest body weight";
run;
```

When there are multiple variable names within a statement, a vertical list can sometimes make things easier to read. You can also use this when you have multiple options in a single statement or multiple arguments in single function.



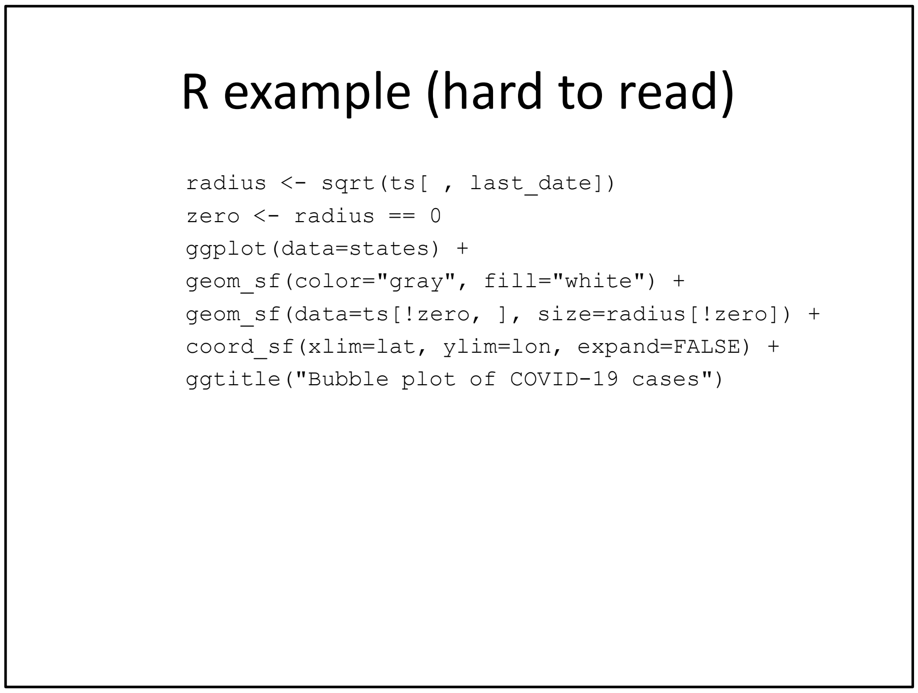
SQL example

```
select
    sex,
    count(*) as number_of_children
  from titanic_table
  where age < 18
  group by sex
```

You can see some of the same principles in this SQL code.

The from, where, and group by statements are indented.

The varibles sex and count(*) are a vertical list and are double indented because they are pare of the select statement.



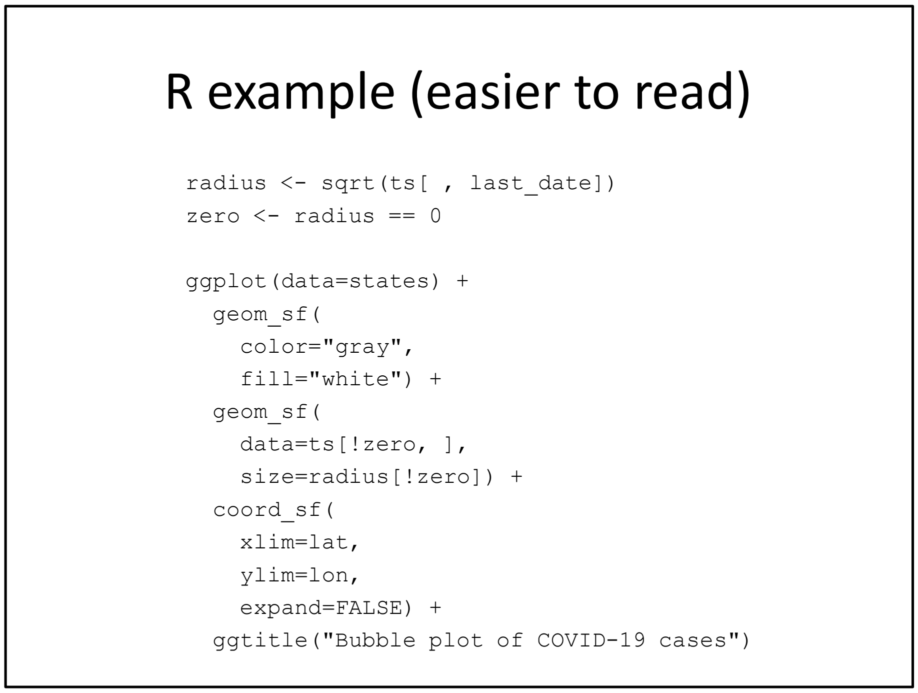
R example (hard to read)

```
radius <- sqrt(ts[ , last_date])
zero <- radius == 0
ggplot(data=states) +
geom_sf(color="gray", fill="white") +
geom_sf(data=ts[!zero, ], size=radius[!zero]) +
coord_sf(xlim=lat, ylim=lon, expand=FALSE) +
ggtitle("Bubble plot of COVID-19 cases")
```

Here's an example using R. It's a bit more advanced, but notice the three commands that create new variables: i, circle_size, and non_zero_cases. They are separated by a blank line from the ggplot function.

The ggplot function has several graphics options that are added: geom_sf (twice), coord_sf, and ggtitle. Each of these is on a separate line and indented consistently.

Each of these graphics options have multiple arguments and they are arranged in a vertical list.



R example (easier to read)

```
radius <- sqrt(ts[ , last_date])
zero <- radius == 0

ggplot(data=states) +
  geom_sf(
    color="gray",
    fill="white") +
  geom_sf(
    data=ts[!zero, ],
    size=radius[!zero]) +
  coord_sf(
    xlim=lat,
    ylim=lon,
    expand=FALSE) +
  ggtitle("Bubble plot of COVID-19 cases")
```

Here’s an example using R. It’s a bit more advanced, but notice the two commands that create new variables: zero and circle_size. They are separated by a blank line from the ggplot function.

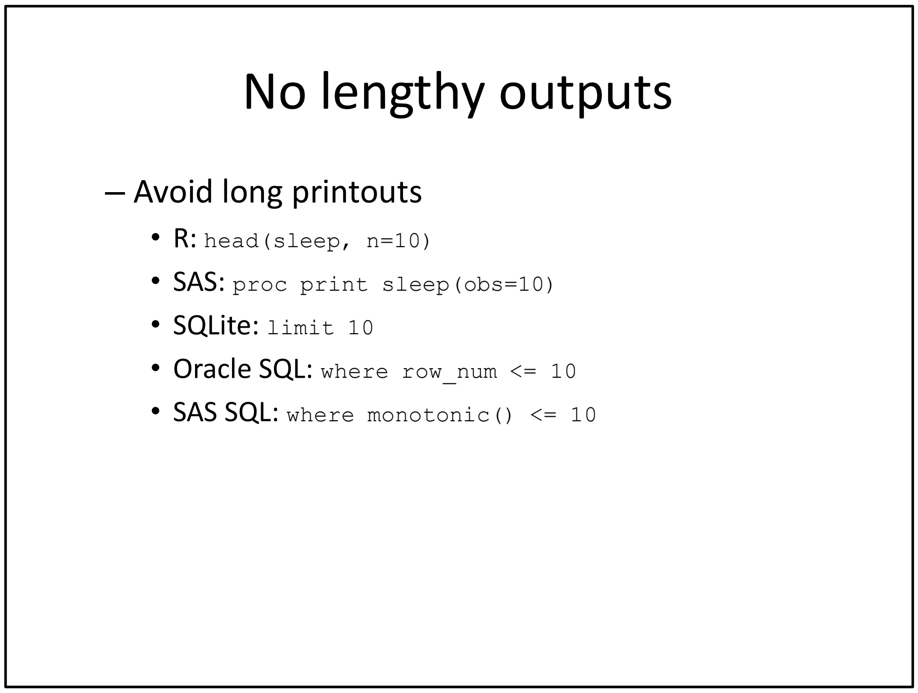
The ggplot function has several graphics options that are added: geom_sf (twice), coord_sf, and ggtitle. Each of these is on a separate line and indented consistently.

Each of these graphics options have multiple arguments and they are arranged in a vertical list. Notice how easy it is now to identify the individual arguments:

color and file for the first geom_sf option

data and size for the second geom_sf option

xlim, ylim, and expand for the coord_sf option



No lengthy outputs

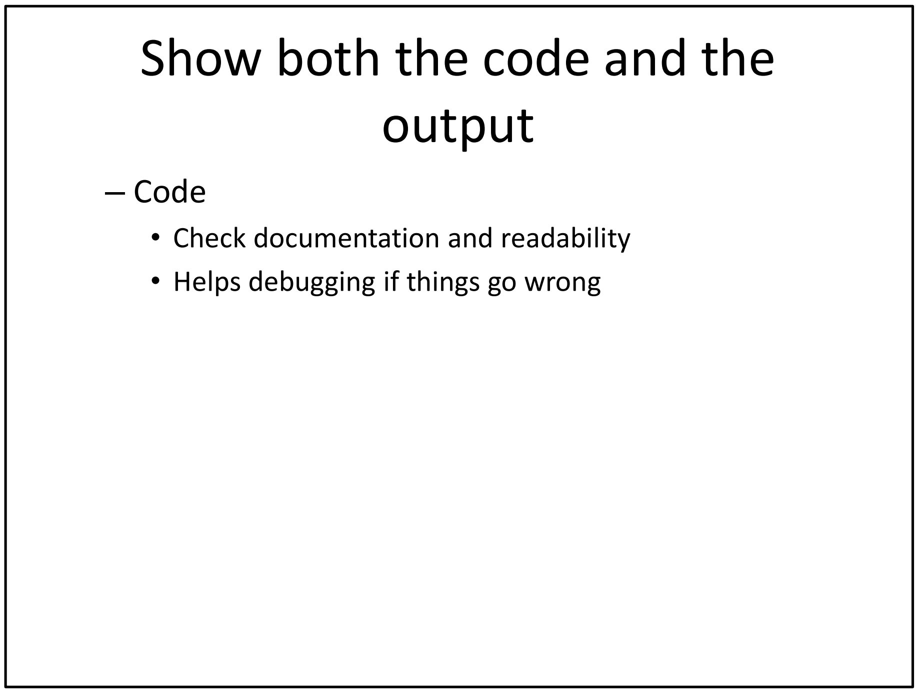
– Avoid long printouts
  - R: `head(sleep, n=10)`
  - SAS: `proc print sleep(obs=10)`
  - SQLite: `limit 10`
  - Oracle SQL: `where row_num <= 10`
  - SAS SQL: `where monotonic() <= 10`

You should never print out all the rows in a lengthy dataset. If I don’t explicitly ask you to limit the number of rows that you print, limit them anyway.

In R, you use the head function and in SAS you include obs= in parentheses after the dataset name.

In SQL it is a bit trickier and depends on which version of Oracle you are using. It is also complicated by a monotonic function that SAS uses when it accesses SQL files.

You’ll see lots of examples of this in various lectures.



Show both the code and the output

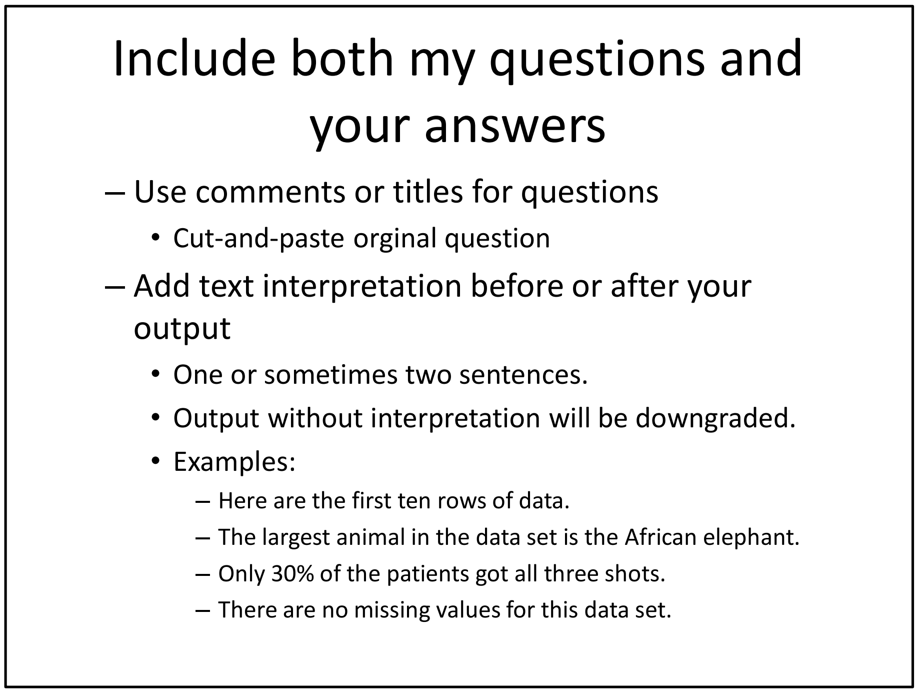
- Code
  - Check documentation and readability
  - Helps debugging if things go wrong

I need to see both your code and your output. Ideally this should be combined in a single pdf document, but you can submit two separate files, one for the code and one for the output.

The code is important because it allows me to assess whether you have a good documentation header and if you are using line breaks, indents, and vertical lists to make your code more readable.

It also helps with debugging if your program doesn't do what it is supposed to do.

You need to include the output, of course, to prove that your program actually ran properly.



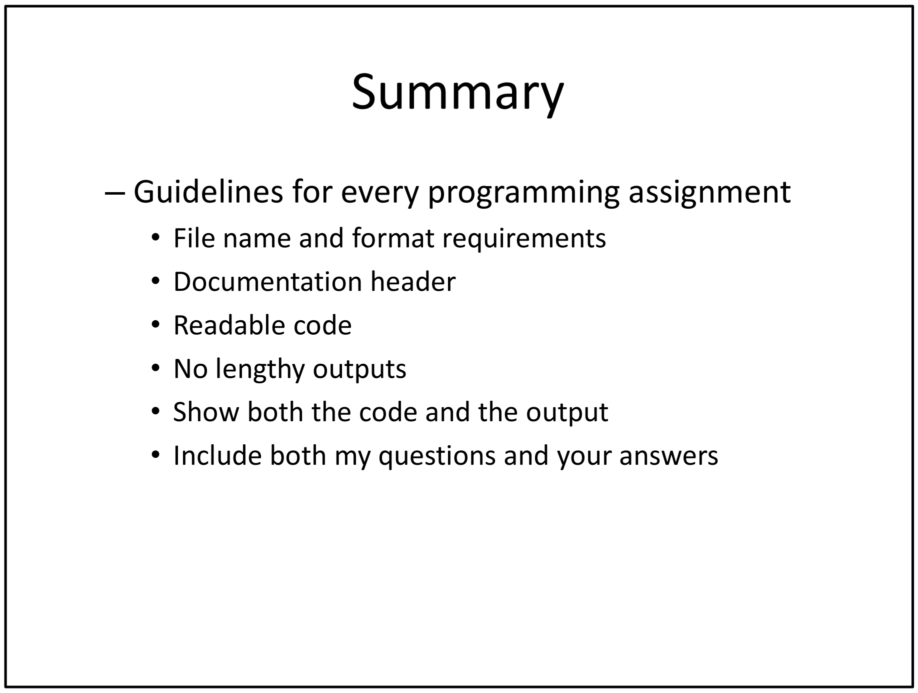
Include both my questions and your answers

- Use comments or titles for questions
  - Cut-and-paste orginal question
- Add text interpretation before or after your output
  - One or sometimes two sentences.
  - Output without interpretation will be downgraded.
  - Examples:
    - Here are the first ten rows of data.
    - The largest animal in the data set is the African elephant.
    - Only 30% of the patients got all three shots.
    - There are no missing values for this data set.

I need to see the original questions in the code as a comment or in the output as a title or footnote. With mutliple assignments to grade in multiple classes, I often fail to remember what I wanted you to do for a particular homework assignment.

As you answer each question, I want a brief comment. Often it is a single sentence, but you might need two or maybe three sentences. What I needs is an explanation of how the output answers the question. This might be obvious, but it still needs to be done. If you include the correct output, but do not add a brief comment, you will lose points on your assignment.

Here are some examples of the comments you might include.



Summary

– Guidelines for every programming assignment
  - File name and format requirements
  - Documentation header
  - Readable code
  - No lengthy outputs
  - Show both the code and the output
  - Include both my questions and your answers

Remember these guidelinesfor every assignment that you turn in. Use the proper file name and format, include a documentation header, make your code readable, avoid lengthy outputs, show both your code and your output and be sure that both the original question and your answer appear in what you submit.