**Lesson 1:**

**curly brackets { } at the start and end of the script**:

Curly brackets are usually used to encapsulate code that is attached to structures like a for loop, if-else statement, or function. In this case, the brackets are encapsulating the whole script. This is used to fix a bug in R that few people know about, a bug that limits how if-else statements can be spaced.

**rm(list=ls());**

**options(show.error.locations=TRUE);**

The first line clears out the **Environmental** (i.e., all variables stored from previous code executions).

The second line adds a line number to the **Console** output for certain errors. R is not that good at pinpointing errors, but this line helps a little.

**Semicolons (;):**

These are used to explicitly tell R where the end of a command is. They are not required in R, but they do help with debugging and are required in other languages like C.

**read.csv() parameters:**

The parameters (e.g., ***file***, ***sep***, ***header***…) of a function are important as they are the knobs that tweak a function call. The more explicit you are at using them, the easier it is to modify them and extend, debug, and tweak your code.

Note: ***read.csv()*** uses the same parameters as ***read.table()*** but sets some of them to different default values. To look more deeply into this type ?read.csv into the Console window in RStudio.

**stringAsFactor in R versions 3 and 4**

***stringAsFactor*** controls how R stores string columns in a data frame. If ***TRUE***, then R stores the columns a factor, if ***FALSE*** then R stores the column as a string. In R 4.0 the default was changed from ***TRUE*** to ***FALSE***.

**GGPlot Components:**

In this workshop, we call the GGPlot plot window the ***canvas*** and the different parts you can put on the canvas the ***components***. A complete (and very large) list of components can be found here:

<https://ggplot2.tidyverse.org/reference/>

**Need help with a specific component:**

1. Make sure you have included the GGPlot package: library(package=ggplot2)
2. Type ?geom\_point (or whatever you want help with) in the **Console** Window
   * If you have not included the package type: ?ggplot2::geom\_point

**Color names:** <https://stat.columbia.edu/~tzheng/files/Rcolor.pdf>

**Point shapes**: <http://www.sthda.com/english/wiki/ggplot2-point-shapes>

**R built-in constants** (from a programmer's perspective, this is an amusingly small list)**:** <https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/Constants>

**Lesson 2:**

**Rstudio suggestions**

In RStudio, when you type in the name of a data frame followed by **$**, RStudio will give you a list of all possible values (i.e., columns), try typing in the script:

abundanceData$

**Subsetting a data frame**

data=abundanceData[year2008,]

means rows given by the ***year2008*** vector and all columns.

**Line types:** <http://www.sthda.com/english/wiki/ggplot2-line-types-how-to-change-line-types-of-a-graph-in-r-software>

**theme\_ components**

There are a LOT of themes in GGPlot – just start typing theme\_ and RStudio will show you the list.

**scale\_x\_continuous() mapping**

You are essentially mapping ***labels*** to ***breaks*** (i.e., ticks) on the x-axis. So, the two values must have the same length – in this case, **12**.

**scale\_x\_continuous() does more than change breaks and labels:**

**scale\_x\_continuous**, and the corresponding **scale\_y\_continuous** are the components you use if you want to create a logarithmic plot (or any other type of scaling)

**The warning message after executing the script:**

In this whole workshop, the warning messages are almost exclusively are because of the **NA**s in the data, which cannot be plotted. We can safely ignore the warnings.

**Lesson 3:**

**color as a mapping and as a subcomponent**

A confusing part about ggplot() is that physical properties like ***color*** can be:

1. a ***mapping*** parameter in ***aes()*** that maps the values of a column to colors on the plot -- this will also create a legend based on the ***color*** mappings
2. a subcomponent of ***geom\_line*** (or any other ***geom\_***) that sets the color of the plot to a specific value – in this case, color is NOT tied to the data.

You never want to use color in both places – they will conflict!

**Setting measurement values**

legend.key.width = unit(3,"cm")

This is a very common structure in GGPlot where instead of directly setting the value of a property (e.g., width) you set the value to an object, ***unit()***, that has the properties you want, ***3cm***.

**Lesson 4:**

**The ~ operator**

There are two distinct roles that **~** play in this lesson:

**First role – axes operator:**

**facet\_wrap(facets = ~year) +**

In this case. the ~ is an axes operator used to modify what is placed on the axes.

The form is ***(y-axis) ~ (x-axis)***.

This is much easier to see using ***facet\_grid()*** instead of ***facet\_wrap()***

Try changing the line to:

facet\_grid(facets = ~year) +

And then change it to:

facet\_grid(facets = year~.) + # yes, you need the dot

The top change facets the plots along the x-axis, the bottom change facets the plots along the y-axis.

The dot (.) sort of translates to "the original value" or "no change" and is only needed when you are changing the y-axis but not changing the x-axis.

**Second role – formula operator:**

**sec.axis = sec\_axis(trans= ~.\*coeff\_WB\_Zoo, # second axis**

In this case, ~ is a formula operator

The form is: **f(x) ~ x**

So, **~.\*coeff\_WB\_Zoo** is the formula **f(x) = x \* coeff\_WB\_Zoo**

Again, the dot (.) sort of translates to "the original, or **x**, value"

Other possible formulas:

sec.axis = sec\_axis(trans= ~.+500, # add 500

sec.axis = sec\_axis(trans= ~./coeff\_WB\_Zoo, # divide by the coef

sec.axis = sec\_axis(trans= ~.^2, # square the values

sec.axis = sec\_axis(trans= ~.\*5+10 # multiply by 5 and add 10

**Comma formatting:**

scale\_y\_continuous(labels = scales::comma) +

***comma*** is a function within the ***scales*** package that formats number to standard notation with a comma. The scales package is a commonly used package for manipulating and reformatting numbers

If you had already included scales in your script: library(package=scales)

Then the line in the script could have been written:

scale\_y\_continuous(labels = comma) + # no need to reference scales

**Lesson 5:**

**group as a mapping parameter:**

***group***, in a sense, is being used to help the handshaking between **GGPlot** and **gganimate**. ***group*** helps GGPlot render the plot in a way that can be easily handed off to the gganimate component. It can be redundant, and I suspect that a few years from now, as gganimate and GGPlot mature, it will not be needed.

**ggplots are variables**

The following code crate a (very large) variable called plot5a:

plot5a = ggplot(data=abundanceData[apr\_to\_sept,])+

geom\_line(mapping=aes(x = month, y = whitesucker,

group = year, color="White Sucker")) +

geom\_line(mapping=aes(x = month, y = zooplankton \* coeff\_WS\_Zoo,

group = year, color="Zooplankton")) + …

***plot5a*** contains plot information and the variable can be seen in the ***Environment*** Window.

Since ***plot5a*** is just save plot information, you can use it as a base for another plot.

# plot5a\_1 includes everything from plot5a

plot5a\_1 = plot5a +

# and adds (or changes) the following

labs(title = "Plot 5a.1",

subtitle = "Zooplankton vs. Larval white sucker abundance: {closest\_state}", # {closest\_state} will refer to the varying state

x = "Month",

y = "Number of Zooplankton",

color = "Species") +

theme\_bw() …

**Plotting variables:**

Saving the plot information to a variable does not create a plot. The command plot(plot5a) creates the plot. It says to take all the plot information from ***plot5a*** and render it in a plotting window.