Introduction to R

R Tutorial

This tutorial will introduce you to the basic coding syntax used in R. The topics include:

- Comments
- Operators
- Data Types
- Creating New Variables
- Complex data types
- Functions
- Packages
- Coding Tips
- Getting Help

Comments

Comments in R code are words, phrases, or sentences that are not executed and describe what the associated R code is doing. Comments are extremely important for your future self to remember what data summary, analysis, or plot you were making. It also helps your instructor grade your assignments. You are expected to use comments extensively within your R code to indicate your name, the assignment you are working, the question you are answering, and a general description of the analysis you are performing. Many of the tutorials and assignments have comments already completed, you must include all comments and additional comments you write.

So, just how do you add comments? Comments are made by using the hashtag, #, character in front of the text. The hashtag ONLY comments the single line.

```
#This is a comment line and will not be executed by R.
```

RStudio uses different colors for executable R code and comments. You can change your color scheme in RStudio under the "Tools"»"Global Options..."»"Appearance" Dialog box.

Operators

R has two basic types of operators, arithmetic and logical. You will use both throughout this semester so it's important to understand the difference.

Arithmetic Operators:

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation

Logical Operators:

Operator	Description
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	exactly equal to
!=	not equal to

Assignment operators

R has two types of assignment operators. These are used to assign values to objects (which you will learn below). However, it's important to know these assignment operators before moving on. <- and = Note these two assignment operators function the same. R has historically used "<-" but you will often see "=" in some tutorials posted online. Note that a single = is **DIFFERENT** than a double ==. The single = is an assignment operator and the double == is a logical operator to determine if two objects are exactly equal.

Data Types

R has several basic data types that you will use. Below is a brief description and R code used to create these data types. Note the use of comments in the R code to describe the data types and the use of a function (more below) that will return the data type of an object. R stores data of similar and dissimilar types within more complex types of containers; vector, matrix, data frame, and list. These will be discussed in more detail after introducing you to creating new variables/objects.

Numeric

Numeric data types can take on any number with decimals. You will often use these with temperature, catch rate, weight, etc.

```
#Assign the value of 20.5 to the object x
x <- 20.5
#print the values within the object x
x
```

```
## [1] 20.5
```

```
#Determine the data type of object x
class(x)
```

```
## [1] "numeric"
```

Integer

Integer data types can take on any whole without decimals. You will often use these with counts of individuals.

```
#Assign the integer value of 6 to the object y
y <- as.integer(6)
#print the values within the object y
y
```

```
## [1] 6
```

```
#Determine the data type of object y
class(y)
```

```
## [1] "integer"
```

Character

Character data types represent string values. You will often use these for captions and column headers.

```
#Assign the character string "Goodson Pond" to the object z
z <- "Goodson Pond"
#print the values within the object z
z
```

```
## [1] "Goodson Pond"
```

```
#Determine the data type of object z
class(z)
```

```
## [1] "character"
```

```
#You can also convert numeric or integer data to characters using a
#special function "as.character()"
#Assign the character string "5.67" to the object v
v = as.character(5.67)
#print the values within the object v
v
```

```
## [1] "5.67"
```

```
#Determine the data type of object v
class(v)
```

```
## [1] "character"
```

Creating New Variables/Objects

R uses "objects" to hold data, plots, and statistical results. So data are stored in objects, R takes the data within an object to run an analysis and the output of the analysis are stored in an object. Objects are specified in R using any string of characters. However, it is customary to use a name that provides a basic description of the data, plot, or statistical results. For example, if you are storing a numeric value for temperature, you can use the object name "temp".

```
#create a new object called "temp" and assign it the value of 18.2
temp <- 18.2
#print the value stored in the object "temp"
temp
```

```
## [1] 18.2
```

Complex data types

This section describes creating vectors, matrices, data frames, and lists.

A vector is a sequence of data of the **SAME** basic data type, data types can't be mixed. A single vector can contain all numeric, all integer, or all character types.

Vectors

```
#create a vector of largemouth bass catch rates by site. Each number represents
#the total number of largemouth bass collected at a single site. The name used
#for the object which contains the total catch is "lmb" which is an
#abbreviation for largemouth bass.
lmb <- c(12,35,66,10,11,8)
#accessing elements within the lmb vector by specifying the index within
#brackets. The following line returns the catch rate at the third site.
lmb[3]
```

```
## [1] 66
```

```
#create a vector of character data
c("one","two","four","two","five","two")
```

```
## [1] "one"  "two"  "four" "two"  "five" "two"
```

Matrix

A matrix is a collection of data elements arranged in a two-dimensional layout. The layout is the same as an Excel spreadsheet. The matrix created in the following code represent catch rates of largemouth bass at six sites (rows) over four years (columns). Each row contains catches from a single site across all years and each column contains catches across all sites for a single year. It's important to note that a matrix will only contain data of the same type.

```
lmb_byrow <- matrix(c(12,16,25,33,
                      35,40,10,45,
                      66,85,33,21,
                      10,10,25,10,
                      11,3,6,18,
                      8,7,12,22),  #the data elements entered one row at a time
                    nrow = 6,      #number of rows
                    ncol = 4,      #number of columns
                    byrow = TRUE)  #fill in the matrix by ROW
lmb_byrow #print the matrix in the console
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   12   16   25   33
## [2,]   35   40   10   45
## [3,]   66   85   33   21
## [4,]   10   10   25   10
## [5,]   11    3    6   18
## [6,]    8    7   12   22
```

```
#Elements of a matrix are accessed by specifying the row then column in
#brackets. To access the catch from the third site in the fourth year use:
lmb_byrow[3,4]
```

```
## [1] 21
```

```
#Access all catches from the third site (row) in the matrix,
#leave the column index empty
lmb_byrow[3,]
```

```
## [1] 66 85 33 21
```

```
#Access all catches from the fourth year (column) in the matrix,
#leave the row index empty
lmb_byrow[,3]
```

```
## [1] 25 10 33 25  6 12
```

```
#This is the same data above but the data are entered by column to
#demonstrate a different way to entering data
lmb_bycolumn <- matrix(c(12,35,66,10,11,8,
                         16,40,85,10,3,7,
                         25,10,33,25,6,12,
                         33,45,21,10,18,22), #the data elements,one column at a time
                       nrow = 6,      #number of rows
                       ncol = 4,      #number of columns
                       byrow = FALSE)  #fill in the matrix by COLUMN
lmb_bycolumn #print the matrix in the console
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   12   16   25   33
## [2,]   35   40   10   45
## [3,]   66   85   33   21
## [4,]   10   10   25   10
## [5,]   11    3    6   18
## [6,]    8    7   12   22
```

Data frame

A data frame is the same basic structure as a matrix (i.e., two-dimensional layout) but it can contain DIFFERENT types of data in each column. Note that column names must be characters. If you use numbers for column names they must be in quotation marks. However, R will automatically add an "X" to the beginning of numbers in quotation marks when creating column names.

```
#Create a data frame with a column for site names and four years of data.
lmb_df <- data.frame(site = c("A","B","C","D","E","F"),
                     "2018" = c(12,35,66,10,11,8),
                     "2019" = c(16,40,85,10,3,7),
                     "2020" = c(25,10,33,25,6,12),
                     "2021" = c(33,45,21,10,18,22))

lmb_df
```

```
##   site X2018 X2019 X2020 X2021
## 1    A    12    16    25    33
## 2    B    35    40    10    45
## 3    C    66    85    33    21
## 4    D    10    10    25    10
## 5    E    11     3     6    18
## 6    F     8     7    12    22
```

```
#Elements of a data frame can be accessed the same way as matrices
#Catch from site C in 2018
lmb_df[3,2]
```

```
## [1] 66
```

```
#All catches from year 2018
lmb_df[,2]
```

```
## [1] 12 35 66 10 11  8
```

```
#You can also return all elements from a single row by specifying the row name
lmb_df$X2018
```

```
## [1] 12 35 66 10 11  8
```

List

A list is a vector containing other objects. A list can contain multiple vectors, multiple matrices, multiple data frames, or any combination. The example below combined one vector, one matrix, and one data frame together in a single list.

```
#See explanation of code for vector, matrix, and data frame above.
lmb <- c(12,35,66,10,11,8)
lmb_byrow <- matrix(c(12,16,25,33,
                      35,40,10,45,
                      66,85,33,21,
                      10,10,25,10,
                      11,3,6,18,
                      8,7,12,22),
                    nrow = 6,
                    ncol = 4,
                    byrow = TRUE)
lmb_df <- data.frame(site = c("A","B","C","D","E","F"),
                     "2018" = c(12,35,66,10,11,8),
                     "2019" = c(16,40,85,10,3,7),
                     "2020" = c(25,10,33,25,6,12),
                     "2021" = c(33,45,21,10,18,22))

#Combine the three objects created above into a single list
lmb_list <- list(lmb, lmb_byrow, lmb_df)

#Access a member of the list using double brackets [[]]
#returns the matrix from the lmb_list
lmb_list[[2]]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   12   16   25   33
## [2,]   35   40   10   45
## [3,]   66   85   33   21
## [4,]   10   10   25   10
## [5,]   11    3    6   18
## [6,]    8    7   12   22
```

Try to return the vector and data frame from the list on your own.

Functions

Functions in R perform a calculation or action to data. Nothing happens in R without functions. In fact you have already been using functions in this tutorial! Many of the code examples above use the "c()" function. It's common to use parentheses next to a function to indicate that it is a function. R functions are often given a name that hints at what they do. The c() function combines elements into a vector, the data.frame() function creates a data frame, etc. Can you find the other functions you have been using above? Below are some useful functions that you will use throughout this class?

A function includes the function name followed by parentheses. The parentheses contains arguments that the function needs. For example, the mean() function requires a vector within the parentheses. We will use coding introduced above to take the mean of a specific rows or columns from a data frame.

```
#First, lets create a data frame with some data
lmb_df <- data.frame(site = c("A","B","C","D","E","F"),
                     "2018" = c(12,35,66,10,11,8),
                     "2019" = c(16,40,85,10,3,7),
                     "2020" = c(25,10,33,25,6,12),
                     "2021" = c(33,45,21,10,18,22))

lmb_df
```

```
##   site X2018 X2019 X2020 X2021
## 1    A    12    16    25    33
## 2    B    35    40    10    45
## 3    C    66    85    33    21
## 4    D    10    10    25    10
## 5    E    11     3     6    18
## 6    F     8     7    12    22
```

```
#Now calculate the average catch at a year and site
#See if you can determine the coding tricks used to specify specific elements
#Determine the average catch in 2018
mean(lmb_df[,2])
```

```
## [1] 23.66667
```

```
#Determine the average catch at site B
#Note that you must use a different function for row mean than column means
rowMeans(lmb_df[2,2:5])
```

```
##    2
## 32.5
```

```
#You can also save the output of a function into a new object
Mean_2018 = mean(lmb_df[,2])
#Print the mean from 2018 in the console
Mean_2018
```

```
## [1] 23.66667
```

Packages

The R core library comes pre-installed with a wide variety of functions to manage, summarize, plot, and analyze data. However, there will be many occasions where you need to use a functions that isn't available in the R core library. Fortunately, R users are also R creators and contribute useful functions in "Packages". These extension packages can be installed using the RStudio GUI (click the Packages tab, then Install) or with the install.packages() function. The following code will install the {MatrixStats} package. This package contains a variety of functions that are useful when using matrices. For example, colMedians() will return the median from all columns and colSds will return the standard deviation from all columns. You'll use these functions throughout the semester.

After installing a package, you have to tell R to load the package before you can use this. This is accomplished using the library() function

```
#install the matrixStats package. You only have to do this once.
install.packages("matrixStats")

#load the matrixStats package. You will have to do this every time you open R.
library(matrixStats)
```

Install the matrixStats package and use the following functions to generate summary statistics for the lmb_byrow matrix created above. Review the help page for the functions for details.

- colMedians() to calculate the median for each column
- colMeans2() to calculate the mean for each column
- colSums2() to calculate the sum of all values in each column

View the help file associated with the functions above using:

```
#load the matrixStats package first
library(matrixStats)
help(colMedians)
```

Coding tips

Below are a few coding tips to remember

- Avoid entering code directly in the console. Code entered in the console can't be easily reused.
- Case matters! "A" is not the same as "a" in R.
- When copying code from a source, make sure you are using a comma "," or period "." when needed. They look similar with some fonts but they do very different things in R.
- Write human readable code. Use space to your advantage. The matrix created above could be accomplished on a single line but the single line would be difficult to read. Don't be afraid to use blank spaces and hard returns to improve the human readability of your code.
- Put space between and around variable names and operators
- Make sure you have the same number of open parentheses "(" as you do closed parentheses ")"
- Make sure you have an even number of quotation marks. If you have an open quote, there must be a closed quote to encapsulate text.
- Use meaningful variable names of one to three words (but the variable names can have spaces!)
- Write complete and helpful comments describing your code
- Keep a consistent style. Avoid using capitalization for some variables but not others.
- Save your work as you go! You can quickly save the R file you are working on by typing "Ctrl + s"
- R will return an error in the console if there is something wrong with your code. The errors in R have become easier to understand than they used to. If there is an error in the spelling of a function, R will tell you it couldn't find the function; if you are referencing the fifth column in a matrix but the matrix only has four columns than R will tell you the subscript is out of bounds. There are many other types of errors that might not be as self explanatory. Always copy the error and paste it in a search engine.
- New users often find the attach() function. –AVOID IT AT ALL COSTS!– The function has good intentions by loading a dataset into memory so you can access a column by only using the column name but it often becomes hard to keep track of what is attached, what variables are available, and what dataset you need.

Getting Help

R provides documentation for each package and function. You can obtain the help file by entering ?c or help(c) and R will locate the help file associated with the "c" function. You can replace "c" with the name of any function.

```
help(c)
```

What if you don't know what the function is for a particular task? A search engine, like Google, is a great place to find the function. Search for a phrase that describe the task you want to do. For example, "How do you calculate the median of all columns in a matrix in R", will return examples from the matrixStats package and other methods. You will quickly see that there are numerous ways to the the same thing in R.