**Lists in R**

# Vectors, Again

Recall that a **vector** is a sequence of entries of the *same data type*. The following are all vectors:

# Create numeric, character and logical vectors   
numerics <- c(1, 2.5, 5, -1, 0.08, 0)  
characters <- c("abc", "d", "6", "six", "l", "f")  
logicals <- c(TRUE, FALSE, FALSE, TRUE, TRUE, TRUE)

At the risk of overemphasis, each of these 3 objects are vectors because the entries of each are of the same data type. The entries of the vector numerics are all numerical (real numbers or integers); the entries of characters are all characters; and the entries in logicals are all logical operators (TRUE/FALSE). To verify, you can check the Environment pane under Values where these vectors are listed with an abbreviated description of their data type (chr, logi, and num). You can also use the class() function, for example:

# Verify data type in characters vector   
class(characters)

# Lists

A data frame generalizes a vector in the sense that it combines vectors of the same length into its own data structure. Another way to generalize a vector is to allow different data types in the sequence of entries. A *list* is a sequence of entries that accommodates a mixture of data types (and much, much more). To create a list, instead of the c() function, use list() around its components:

# Create a list   
my\_list <- list(7, "Doom", FALSE, -98, TRUE, "ALL CAPS!")

All 3 data types are present in my\_list, so it’s certainly not a vector. Moreover, note that, upon running the command, R adds my\_list under Data in the Environment pane. Clicking the blue-circle-with-white-arrow icon displays the data types of each of its contents.

As intimated in the (non-exhaustive) definition given above, lists are not limited to sequences of entries. List components may include data frames, matrices, results from statistical analyses, vectors, lists (and sublists of lists) and much more. The vast range of objects that are found in lists is why we use the word components, instead of entries. To illustrate the point, run these commands and examine the structure of my\_crazy\_list in the Environment pane:

# Create my\_crazy\_list  
doom <- "ALL CAPS when you spell the man's name!"  
num\_vector <- c(12, 42, -12)  
df <- data.frame(characters, logicals)  
my\_crazy\_list <- list(doom, num\_vector, df, list(doom, df))  
my\_crazy\_list

It’s complicated. To temper this type of complexity it’s often wise to name a list’s components, which then allows you to access components by name. Assigning names to components also clarifies the structure of a list and, consequently, makes it less likely you commit coding errors. Modify my\_crazy\_list to include names for its 4 list components:

# Add names to list components of my\_crazy\_list   
my\_crazy\_list <- list(d = doom, n = num\_vector, f = df, l = list(doom, df))  
my\_crazy\_list

## Accessing (Indexing) Elements of Lists

On account of their greater complexity, accessing components of lists is more complicated than subsetting vectors. To understand the component parts of a list, call the structure function, str(), as follows:

# Examine structure of my\_list   
str(my\_list)

A key difference between indexing vectors and indexing lists concerns the use of the square brackets ([]). With a vector, calling, for example, x[7] returns the seventh element of the vector. But the command my\_list[4] returns a list, and specifically a list containing one element -98. To extract the element -98 and just the element, use double square brackets: my\_list[[4]]. You should compare the results from the following commands to verify you understand the difference between the use of [] and [[]] in lists.

# Accessing the 3rd and 4th elements of my\_list as sublists  
my\_list[3]  
my\_list[4]

# Extracting the 3rd and 4th elements of my\_list as entries  
my\_list[[3]]  
my\_list[[4]]

So, the double-square brackets, [[]], extract a list component and return it to its original data-type/data-structure. For example, if the list component is a vector, then using [[]] will return a vector. There are in fact 3 ways to accomplish this. To verify, let’s return to my\_crazy\_list and confirm that these 3 commands return the same object:

1. my\_crazy\_list[[3]]
2. my\_crazy\_list[["f"]]
3. my\_crazy\_list$f

Each of these is useful in different contexts. But remember that qualifying phrase: “return it to its original data-type/data-structure.” If you use single brackets, you’ll always get a list back.

You can exclude an element by using a negative index. For example, my\_list[-5] returns a sublist of all elements from my\_list *except* the 5th element.

## So What?

Lists are important R data structures because:

1. They allow you to organize and recall disparate information in a relatively simple way.
2. Results from many R functions return lists, and thus to work with these lists you’ll need to know how to extract their components.

An important example to illustrate the second point is the summary() function, which returns, among other things, standard regressions results. For a regression model, summary() is a list of these results. Consider this contrived example and be sure to observe the output from the last command:

# Regressing mpg on weight   
mtcars\_reg <- lm(mpg ~ wt, data = mtcars)  
  
# Get regression summary  
summary(mtcars\_reg)  
  
# View structure of summary(mtcars\_reg)  
str(summary(mtcars\_reg))

From that last command you should see a sublist of summary(mtcars\_reg) called coefficients containing estimates as well as - and -values. You’ll often need to extract coefficient estimates and their standard errors and, thus, you’ll unavoidably have to apply the list rules from above. Let’s look at the coefficients sublist:

# View estimates from mtcars\_reg  
summary(mtcars\_reg)$coefficients

## Estimate Std. Error t value Pr(>|t|)  
## (Intercept) 37.285126 1.877627 19.857575 8.241799e-19  
## wt -5.344472 0.559101 -9.559044 1.293959e-10

To extract any entry in this $a$ $2 \times 2$ array, use square brackets, [], with 2 indicies within the square brackets: The first index referring to the row of the array, the second referring to the column of the array. If you’ve dealt with matrices before, this way to specify an entry of a matrix should sound familiar. That is, the entry in the row and column from matrix is denoted by ; the row index preceding the column index. Likewise with lists: To extract the entry in the the row and column of a list, you’ll enter [i,j]. So you can think of square brackets as a type of matrix.

Let’s consider some examples. To extract the slope estimate, wt, which is located in the 2nd row and the first column, run:

# Extract slope estimate  
summary(mtcars\_reg)$coefficients[2,1]

What about its standard error?

# Extract standard error of slope estimate  
summary(mtcars\_reg)$coefficients[2,2]

Dividing a coefficient’s estimate by its standard error approximates its -value. Let’s verify for wt:

# Compute t-value of wt coefficient  
summary(mtcars\_reg)$coefficients[2,1]/summary(mtcars\_reg)$coefficients[2,2]

This is a lot of typing with little payoff. You’ll learn how to expedite the process by creating your own function that will extract estimates with much less effort.