Week 3

Discuss this week's module: Week 3: Loop Functions and Debugging.

Week 3

[Tips] Understanding lapply()

Al Warren Mentor Week 3 · 6 months ago · Edited

The lapply function takes a vector object and passes each item in the vector through a function. Each time it passes an item to that function, it stores the return value of that function in a list item. After all items have been processed, it returns the list.

Suppose I have an integer vector:

```
vectorA <- 1:6
vectorA
## [1] 1 2 3 4 5 6
```

Lets use lapply to find all the square roots:

```
listA <- lapply(vectorA, sqrt)

listA
## [[1]]
## [1] 1
##
## [[2]]
## [1] 1.414214
##
## [[3]]
## [1] 1.732051
##
## [[4]]
## [1] 2
##
## [[5]]
## [1] 2.236068
##
## [[6]]
```

```
20  ## [1] 2.44949
```

That worked out pretty well. But lapply returned a list and I want a vector. How do I do that?

ListA is a simple list containing the square roots of the original vector. One thing we can do with a list is convert it into a simple vector using unlist. Given a list structure x, unlist simplifies it to produce a vector which contains all the atomic components which occur in x.

For example:

```
1  unlist(listA)
2  ## [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490
```

Sometimes, we can shorten things by wrapping lapply in unlist:

```
1  unlist(lapply(vectorA, sqrt))
2  ## [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490
```

What about something more complicated? Like a list of data frames:

```
 1  listB <- list(data.frame(x=1:3, y=1:3*2), data.frame(x=4:6, y=4:6*2))
 2  listB
 3  ## [[1]]
 4  ##   x y
 5  ## 1 1 2
 6  ## 2 2 4
 7  ## 3 3 6
 8  ##
 9  ## [[2]]
10  ##   x  y
11  ## 1 4  8
12  ## 2 5 10
13  ## 3 6 12
14
15  class(listB[[1]])
16  ## [1] "data.frame"
```

What happens if I use lapply to sum all the values in the list?

```
1  lapply(listB, sum)
2  ## [[1]]
3  ## [1] 18
4  ##
5  ## [[2]]
6  ## [1] 45
```

That's not what I expected. How about taking the square roots?

```
 1  lapply(listB, sqrt)
 2  ## [[1]]
 3  ##          x        y
 4  ## 1 1.000000 1.414214
 5  ## 2 1.414214 2.000000
 6  ## 3 1.732051 2.449490
 7  ##
 8  ## [[2]]
 9  ##          x        y
10  ## 1 2.000000 2.828427
11  ## 2 2.236068 3.162278
12  ## 3 2.449490 3.464102
```

Still not what I'm looking for. How about something more specific. How about the square roots of x? Wait, it's a list of data frames. How do we aggregate values from a data frame column? How about using an anonymous function:

```
lapply(listB, function(data) sqrt(data$x))
## [[1]]
## [1] 1.000000 1.414214 1.732051
##
## [[2]]
## [1] 2.000000 2.236068 2.449490
```

Remember how we said each item in the list gets passed through the function? Here our example list is a list of data frames. So, when we use lapply, each data frame gets passed through the function and it's return value is collected in a list. In the above function, each data frame is passed as the *data* argument of the function.

But suppose instead of squaring each value, we wanted to sum ALL the values. Instead of using sum in lapply, we just return the column we're interested in. Then we take the results of unlist and sum that.

```
unlist( lapply(listB, function(data) data$x) )
## [1] 1 2 3 4 5 6

sum( unlist( lapply(listB, function(data) data$x) ) )
## [1] 21
```

One last thing. Suppose we need to use a variable column name. That's fine, we just have to use double brackets instead of the $ operator.

```
my_column <- "x"

unlist( lapply(listB, function(data) data[[my_column]]) )
## [1] 1 2 3 4 5 6

sum( unlist( lapply(listB, function(data) data[[my_column]]) ) )
## [1] 21
```

Finally, I used lapply in the examples but you could also use sapply in which case you wouldn't need unlist.

Now, think about the above concepts and how you might use them in some of the assignments. Like reading multiple files and doing something with the data.

Note: This post is now pinned and available to all sessions but has been closed for comments. If you have questions on any of these methods please post a new thread. Thanks.