



ITERATION AND LIST COLUMNS

Jeff Goldsmith, PhD
Department of Biostatistics

Why iterate

- You will frequently encounter problems where you need to do the same basic thing **a lot**
- The “don’t write the same code more than twice” rule motivates the use of functions
- The need to do the same thing a lot motivates formal structures for iterating

for loops

- Loops are the easiest place to start
- Loops consist of an output object; a sequence to iterate over; the loop body; and (optionally) an input object
- It's often handy to keep track of inputs and outputs using lists, given their flexibility

for loops

- The basic structure is:

```
input = list(...)  
output = list(...)
```

```
for (i in 1:n) {
```

```
    output[[i]] = f(input[[i]])
```

```
}
```

Loop functions

- The loop process (supply input vector / list; apply a function to each element; save the result to a vector / list) is really common
- For loops can get a little tedious, and a little opaque
 - Have to define output object and iteration sequence
 - Need to make sure loop body is indexed correctly
 - Often unclear on a first glance exactly how inputs are connected to outputs
- Loop functions are a popular way to clean up loops
 - We'll focus on `purrr::map()`
 - Base R has `lapply()` and similar functions



map

- Goal of map is to clarify the loop process
- The basic structure is

```
output = map(input, f)
```

- This produces the same result as the for loop, but emphasizes the input and function and reduces the amount of overhead
 - Doesn't speed code up (as long as you have well-written loops)
 - Benefit comes from clarity

map variants

- By default, map takes one input and will return a list
- If you know what kind of output your function will produce, you can use a specific map variant to help prevent errors and simplify outputs:
 - `map_db1`
 - `map_lgl`
 - `map_df`
- If you need to iterate over two inputs, you can use map variants to give two input lists / vectors:
 - `map2`
 - `map2_db1`
 - `map2_df`

Process

- I often don't jump straight to a function definition with a map statement to do iterative processes
- One workflow I use is
 - Write a single example for fixed inputs
 - Embed example in a for loop
 - Abstract loop body to a function
 - Re-write using a map statement
- This helps make each step clear, prevents mistakes, and only adds complexity when I need it
- Eventually you'll get used to writing functions and mapping directly

Lists

- In R, lists provide a way to store collections of arbitrary size and type
 - You can mix character vectors, numeric vectors, matrices, summaries...

```

> list(a = rnorm(10), b = c("Jeff", "Goldsmith"), c = summary(runif(100)))
$a
 [1] -0.45570641  1.07079885  0.23944031  0.61202840 -0.09985825 -0.61119970  0.11551818 -0.83438686
 [9]  1.33986752  0.66033877

$b
 [1] "Jeff"      "Goldsmith"

$c
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.01796 0.30540 0.47852 0.49379 0.70405 0.98868
    
```

Data frames

- Data frames, which we've used extensively, are a special kind of list
 - Each list entry is a vector with the same length
 - You can still mix variable classes
 - Printed as a table

```
> data_frame(  
+   a = rnorm(4),  
+   b = c("my", "name", "is", "jeff"),  
+   c = sample(c(TRUE, FALSE), 4, replace = TRUE)  
+ )  
# A tibble: 4 x 3  
       a      b      c  
   <dbl> <chr> <lgl>  
1  0.9609689 my  TRUE  
2  0.9383835 name TRUE  
3 -2.8595221 is  FALSE  
4 -0.6573009 jeff FALSE
```

List columns

- Lists can contain almost anything
 - A list can even contain a list!
- What if an entry in your list is a list, but it has the same length as the other entries?
- Could that be a “column” in a data frame?

List columns

- Lists can contain almost anything
 - A list can even contain a list!
- What if an entry in your list is a list, but it has the same length as the other entries?
- Could that be a “column” in a data frame?

YES!!

List columns

- Lists can contain almost anything
 - A list can even contain a list!
- What if an entry in your list is a list, but it has the same length as the other entries?
- Could that be a “column” in a data frame?

YES!!!!!!



Seriously?

YES!!!!!!

- List columns turn out to be very useful
- Imagine you have an **input list** in a data frame
- You can map a **function** to each element of that **input list**, export the **output list**, and save it in the same data frame
- Keeping everything in one data frame with **list columns** means there are fewer things to worry about

But wait – there’s more!!

- Imagine you have granular data nested within large units
 - Make a **list** storing your granular data table
 - Add the granular data table **list** to a data frame containing data on larger units
- Why stop there??
 - You can store more complex R objects, like output from regressions on each granular data table, in a **list**
 - You can add that **list** to your data frame
- Keeping everything in one data frame with **list columns** means there are fewer things to worry about