# Control Flow: Loops

We want to look at how to control the execution of our code. The three main things we are looking at here are

- `if/then/else` logic and syntax
- looping to repeatedly execute code
- vectorized functions for improved efficiency

This section looks at how to do loops (repeated execution of code) in R.

## Looping in R

There are a number of ways to do looping in R

- `for()`
- `while()`
- `repeat()`

The idea of each is to run some code repeatedly; often changing something with each execution of the code.

### For Loops

The syntax for a `for` loop (most commonly used loop in R) is

```
for(index in values){
  code to be run
}
```

where

- index defines 'counter' or variable that varies
- 'values' define which values index takes on

For example, our index below is `i` and the values it can take on are the integers from 1 to 10 (`1:10`)

```
for (i in 1:10){
  print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
```

```
[1] 10
```

The values don't need to take on numbers and the object you use for the index can be changed:

```
for (index in c("cat","hat","worm")){
  print(index)
}
```

```
[1] "cat"
[1] "hat"
[1] "worm"
```

Of course, the idea is to use the changing values in some meaningful way. Here is a quick example of printing out a particular string based on inputs.

Create two vectors of length 5.

```
words<-c("first", "second", "third", "fourth", "fifth")
data <- runif(5)
```

- Loop through the elements of these and print out the phrase

"The (#ed) data point is (# from data vector)."

- To put character strings together with other R objects (which will be coerced to strings) we can use the `paste()` function. Checking the help we see:

`paste (..., sep = " ", collapse = NULL, recycle0 = FALSE)`

where `...` 'is one or more R objects, to be converted to character vectors.' and the `sep =` argument determines the value by which to separate these objects.

```
paste("The ", words[2], " data point is ", data[2], ".", sep = "&"
```

```
[1] "The &second& data point is &0.348438310204074&."
```

```
paste("The ", words[1], " data point is ", data[1], ".", sep = "")
```

```
[1] "The first data point is 0.050788925262168."
```

Note: `sep = ""` is equivalent to using the `paste0()` function.

Ok, let's put this into a loop!

```
for (i in 1:5){
  print(paste0("The ", words[i], " data point is ", data[i], "."))
}
```

```
[1] "The first data point is 0.050788925262168."
[1] "The second data point is 0.348438310204074."
[1] "The third data point is 0.820016274927184."
```

```
[1] "The fourth data point is 0.105122385779396."
[1] "The fifth data point is 0.383394422242418."
```

- As `i` iterates from 1 to 5, we pull out the corresponding elements of `words` and `data` to make our sentence!

A more useful example would be finding summary statistics about different numeric columns of a data frame (recall this is a 2D structure we often use to store datasets).

- Consider a dataset on batting of Major League Baseball (MLB) players.
    - You may need to run `install.packages("Lahman")` once on your machine before you can run this code

```
library(Lahman)
```

```
Warning: package 'Lahman' was built under R version 4.1.3
```

```
my_batting <- Batting[, c("playerID", "teamID", "G", "AB", "R", "H
head(my_batting)
```

```
   playerID teamID  G  AB  R  H X2B X3B HR
1 abercda01    TRO  1   4  0  0   0   0  0
2  addybo01    RC1 25 118 30 32   6   0  0
3 allisar01    CL1 29 137 28 40   4   5  0
4 allisdo01    WS3 27 133 28 44  10   2  2
5 ansonca01    RC1 25 120 29 39  11   3  0
6 armstbo01    FW1 12  49  9 11   2   1  0
```

- Let's say we want to find the `summary()` for each numeric column of this data set.

```
summary(my_batting[ , "G"])
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   1.00   12.00   34.00   50.74   79.00  165.00 
```

```
summary(my_batting[ , "AB"])
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    0.0     4.0    46.0   139.2   224.0   716.0 
```

That's fine but we want to do it for all the numeric columns. Let's use a for loop!

```
dim(my_batting)
```

```
[1] 108789      9
```

We could do a loop that takes on values of `3:9` (or programmatically `3:dim(my_batting)[2]`).

```
for (i in 3:dim(my_batting)[2]){
  print(summary(my_batting[ , i]))
}
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   1.00   12.00   34.00   50.74   79.00  165.00 
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    0.0     4.0    46.0   139.2   224.0   716.0 
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   0.00    0.00    4.00   18.48   27.00  198.00 
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   0.00    0.00    8.00   36.39   56.00  262.00 
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  0.000   0.000   1.000   6.202   9.000  67.000 
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  0.000   0.000   0.000   1.247   1.000  36.000 
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   0.00    0.00    0.00    2.85    2.00   73.00 
```

Alternatively, the `seq_along()` function can be useful. This looks at the length of the object and creates a sequence from 1 to that length. Remember that a data frame is truly a list of equal length vectors (usually). The length of a list is number of elements. Here that is the number of columns!

```
length(my_batting)
```

```
[1] 9
```

```
seq_along(my_batting)
```

```
[1] 1 2 3 4 5 6 7 8 9
```

Now we can just remove the 1st and 2nd entries of that vector (as they are not numeric columns) and use that as our values to iterate across.

```
for (i in seq_along(my_batting)[-1:-2]){
  print(summary(my_batting[ , i]))
}
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   1.00   12.00   34.00   50.74   79.00  165.00 
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    0.0     4.0    46.0   139.2   224.0   716.0 
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   0.00    0.00    4.00   18.48   27.00  198.00 
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   0.00    0.00    8.00   36.39   56.00  262.00 
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  0.000   0.000   1.000   6.202   9.000  67.000 
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  0.000   0.000   0.000   1.247   1.000  36.000 
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   0.00    0.00    0.00    2.85    2.00   73.00 
```

We likely don't enjoy this format. Although we'll see much easier ways to deal with this, let's initialize a data frame to store our results in. We can initialize the type of data to store in a particular column using `character()`, `numeric()`, `logical()`, etc.

```r
summary_df <- data.frame(stat = character(),
                         min = numeric(),
                         Q1 = numeric(),
                         Median = numeric(),
                         Mean = numeric(),
                         Q3 = numeric(),
                         Max  = numeric())
summary_df
```

```
[1] stat   min    Q1     Median Mean   Q3     Max   
<0 rows> (or 0-length row.names)
```

Ok, now let's fill this in as we loop (note we use `i-2` to start filling in at row 1 and we grab the statistic we are summarizing from the `colnames` of the `my_batting` data frame).

```r
for (i in seq_along(my_batting)[-1:-2]){
  summary_df[i-2, ] <- c(colnames(my_batting[i]),
                         summary(my_batting[ , i]))
}
summary_df
```

```
  stat min Q1 Median             Mean  Q3 Max
1    G   1 12     34 50.7404884685033  79 165
2   AB   0  4     46  139.24132035408 224 716
3    R   0  0      4  18.483495574001  27 198
4    H   0  0      8 36.3886054656261  56 262
5  X2B   0  0      1 6.20202410170146   9  67
6  X3B   0  0      0 1.24707461232294   1  36
7   HR   0  0      0 2.85015029093015   2  73
```

## While Loops

- Should know about these too!

```r
while(cond) {
    expr
}
```

- If `cond` is `FALSE` then the loop never executes.
- We won't use these much.

## Other Loop Things

- Sometimes we need to jump out of a loop. `break` kicks you out of the loop.

```r
for (i in 1:5){
```

```
          if (i == 3) break #can put code to execute on the same line
          print(paste0("The ", words[i], " data point is ", data[i], "."))
        }
```

```
[1] "The first data point is 0.050788925262168."
[1] "The second data point is 0.348438310204074."
```

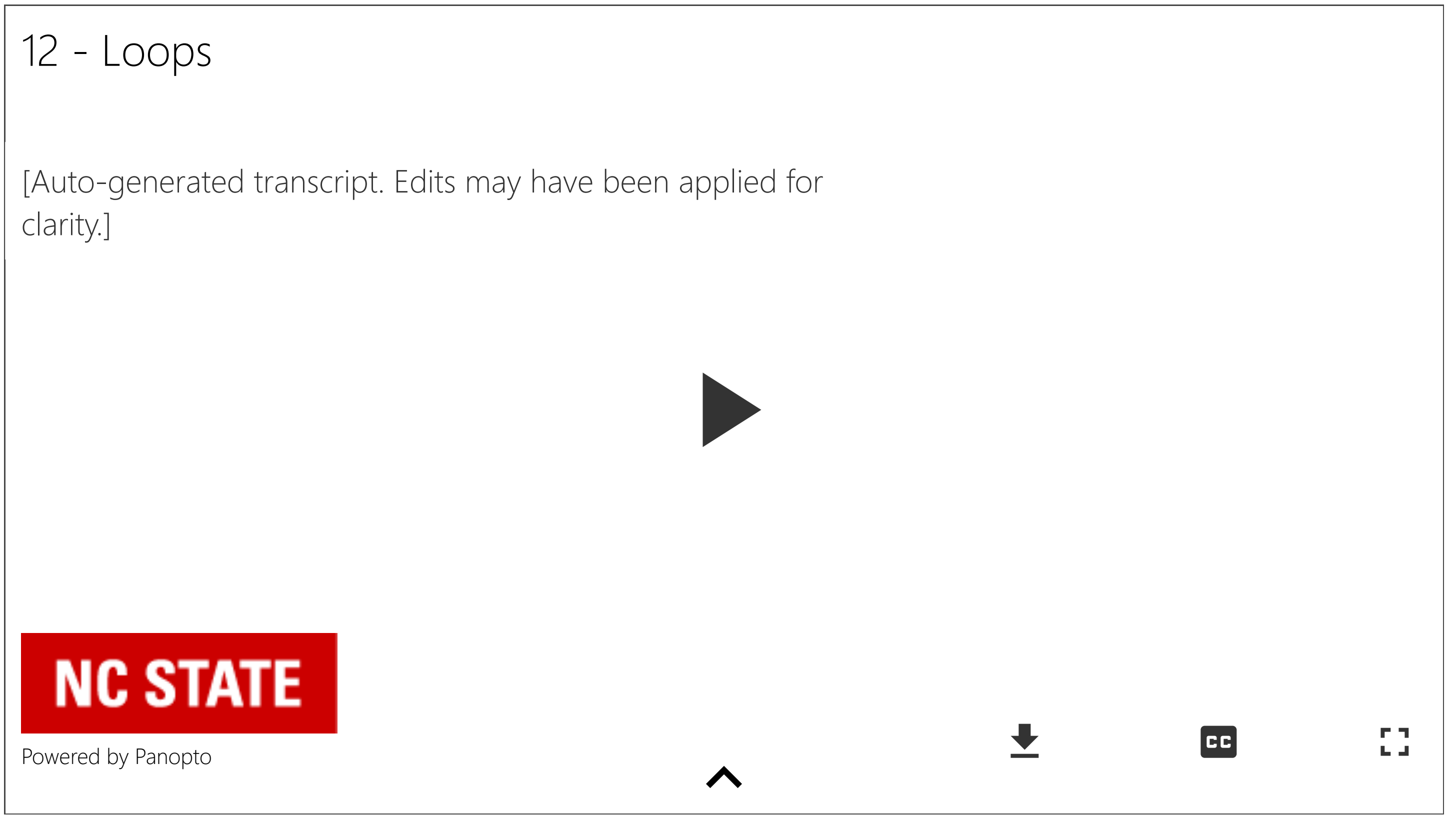
- Sometimes we need to skip an iteration. `next` jumps to the next iteration of the loop.

```
        for (i in 1:5){
            if (i == 3) next
          print(paste0("The ", words[i], " data point is ", data[i], "."))
        }
```

```
[1] "The first data point is 0.050788925262168."
[1] "The second data point is 0.348438310204074."
[1] "The fourth data point is 0.105122385779396."
[1] "The fifth data point is 0.383394422242418."
```

## Quick R Video

Please pop this video out and watch it in the full panopto player!


12 - Loops

[Auto-generated transcript. Edits may have been applied for clarity.]

## Recap!

- Loops provide a mechanism to run the same code repeatedly

```
        for(index in values){
          #code to evaluate
        }
```

- index is the variable that changes during each iteration
- values are the values the index takes on