Today we will take a look at repeating the same or similar commands over and over – iterating.

**99 Bottles of Beer**

This example comes from an exercise in the iteration chapter of the first edition of your textbook. They seem to have removed it from the second edition. Remember the song “99 Bottles of Beer on the Wall?” It demonstrates a situation in which we repeat ourselves over and over again, only changing a little bit.

1. Recall what you learned in the last assignment about functions. What does the following function do?

*bottles <- function(n=99){*

*songstring <- paste(n,"bottles of beer on the wall,",*

*n, "bottles of beer. Take one down, pass it around,",*

*n-1, "bottles of beer on the wall.")*

*print(songstring)*

}

This function returns a line of the song as a string based on the value of n passed to it with a default value of 99.

1. Suppose we want to sing this song to young kids. We may want to talk about bottles of soda or bottles of milk. Modify the above function to take a second argument to allow for any liquid we choose. Paste your code here, and the output from calling the function.

bottles <- function(n=99, liquid="beer") {

songstring <- paste(n, "bottles of", liquid, "on the wall,",

n, "bottles of", liquid, ". Take one down, pass it around,",

n-1, "bottles of", liquid, "on the wall.")

print(songstring)

}

>bottles(,"milk")

[1] “99 bottles of milk on the wall, 99 bottles of milk on the wall, 99 bottles of milk . Take one down, pass it around, 98 bottles of milk on the wall.”

1. Now, we want to sing the entire song. Examine the following code and summarize what it does below (do not paste all the output here, it’s huge):

*for (i in 99:1) {*

*bottles(i)*

*}*

Decrements i from 99 to 1 by means of a for loop.

There are several parts to this code. First, the index variable i, then the list that the index variable varies over, then the code that will be repeated. When this code is run, i starts at the first value in the list, then the next, etc., until the whole list has been traversed. Most of the time, the list starts at 1 and continues to some large value, but in this case, we are going backwards from 99 to 1. Finally, there is the code that will be repeated. This code uses i as an input, so the output is different for each cycle through the loop. If i was not referenced, we would get repeats of the exact same output.

1. Use a for loop to print the word “blah” ten times. Paste your code here.

for(i in 10:1){

print("blah")

}

**Simulating a Fair Coin Flip (An example of repeating actions)**

Imagine flipping a fair coin. There are only two outcomes -- heads or tails (discounting weird outcomes like the coin being lost or landing on its edge).

We'd like to simulate that on the computer.

*#Simulate a Fair Coin Flip*

*flips <- sample(c("H","T"), 100, replace=TRUE)*

*flips*

*table(flips)*

5. We have simulated flipping a coin 100 times. How many times would you expect to see heads? Is the observed value (the number of heads you got) equal to the expected value (the number of heads you expected)? (The output of the table command lists each outcome in the first row and the number of times that outcome occurred in the second row.)

Expected: 50h 50t

Actual: 61h 39t

6. Look at the “sample” command. There are three arguments. What do each of the arguments do? What happens if the argument “replace=TRUE” is missing?

Arg1 seems to declares the possible outcomes, arg2 declares sample size, and arg3 removes each item from the set after it is drawn. I assume the unused prob parameter has a default value of 0.5.

Error in sample.int(length(x), size, replace, prob) :  
 cannot take a sample larger than the population when ‘replace = FALSE’

7. What if we want to simulate an unfair coin? Suppose the coin is 90% likely to show heads, and only 10% likely to show tails.

*#Simulate an unfair Coin Flip*

*flips <- sample(c("H","T"), 100, replace=TRUE, prob=c(0.9,0.1))*

*flips*

*table(flips)*

How many heads did you get this time? What did the addition of the fourth argument do?

H: 92 T: 8

Added the probability parameter.

In both of the above cases, you probably got an observed value that is not the same as the expected value. In the first case, you expected 50 heads, and in the second case, you expected 90 heads. Suppose we want to repeat these flips over and over to get a good idea of what the frequency of different outcomes will be. Looping allows that:

*heads <- function(pr=.5){*

*flips <- sample(c("H","T"), 100, replace=TRUE, prob=c(pr,1-pr))*

*print(table(flips)[[1]])*

*}*

*headslist <- vector("integer",1000)*

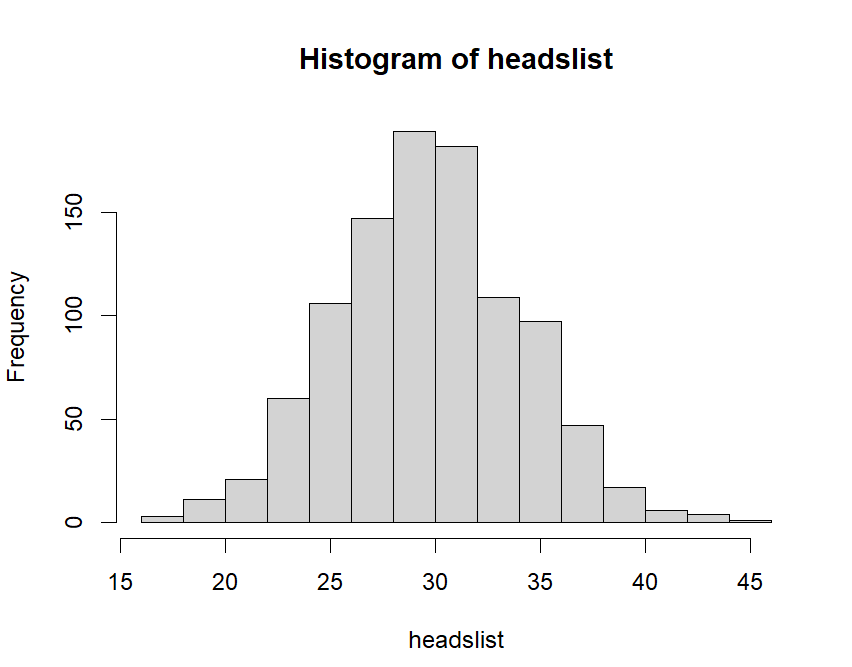
*for (i in 1:1000){*

*headslist[i] <- heads(0.3)*

*}*

*hist(headslist)*

8. Run the above code and copy your graph here. (I did not use ggplot2, but of course you can. It requires a data frame rather than a vector, though.)



9. What does the double bracket in the table(flips)[[1]] command do? What would table(flips)[[2]] give? What about table(flips)[1]?

It appears to extract the 1st item of the table, and (flips)[[2]] gives the 2nd item respectively. Without the brackets it returns the entire H vector.

Note that for the loop above (in 8), I defined a vector of the right size, then filled the vector inside the loop. You can also create an empty vector and add to it.

*headslist <- c()*

*for (i in 1:1000){*

*headslist <- c(headslist, heads(0.3))*

*}*

*hist(headslist)*

This adds the new value to the end of the list. It has the advantage of not needing to know how long the list will be up front, but is slower.

10. Modify the code just above to create a list of 5000 values of heads(0.9) and plot a histogram. Paste your code and your graph here.

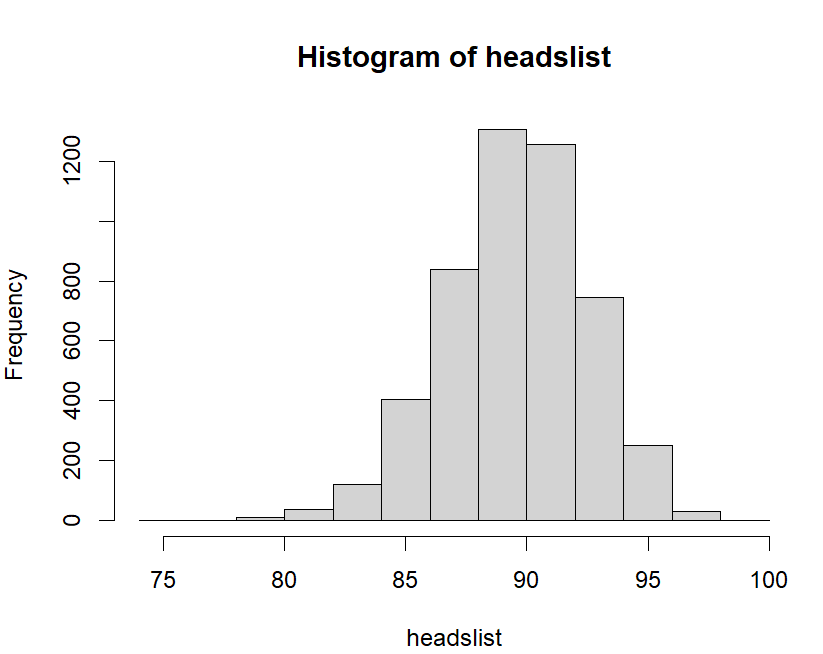
headslist <- c()

for (i in 1:5000){

headslist <- c(headslist, heads(0.9))

}

hist(headslist)



There are other looping options (while, repeat) in R, but they are not used as much as the for loop. The R language has a lot of functions that work as if they are looping on an input. One example of this is replicate.

**Simulating the Roll of Two Dice (Another example of repeating actions)**

*#simulate rolling two fair dice*

*sample(1:6, size = 2, replace = TRUE, prob = rep(1/6, 6))*

*#simulate rolling two fair dice 100 times*

*dice2<-replicate(100,sample(1:6, size = 2, replace = TRUE, prob = rep(1/6, 6))) #replicate does the sampling again and again*

*dice2 #Note that dice2 is a matrix with 2 rows and 100 columns*

11. Run the above code. Notice that replicate has repeated the code inside it. Modify the code to roll 5 dice 20 times. Do not redefine the dice2 variable, we will use it again. Paste your new code here.

sample(1:6, size = 5, replace = TRUE, prob = rep(1/6, 6))

dice2<-replicate(20,sample(1:6, size = 2, replace = TRUE, prob = rep(1/6, 6))) #replicate does the sampling again and again

dice2

12. Suppose after we roll 2 dice, we want to add their values together. We could do that with a loop. Explain how. (Writing code is good, pseudocode is fine also, or just explaining what to do.)

I would probably just create a 3rd vector to add the sum of the dice rolls which would be calculated in each loop. Something like:

dice2 <- replicate(100, sample(1:6, size = 2, replace = TRUE, prob = rep(1/6, 6)))

sums <- numeric(100)

for(i in 1:ncol(dice2)){

sums[i]<- sum(dice2[,i])

}

13. But R has a function that repeats the same command on every row or column of a matrix.

*sumlist <- apply(dice2,2, sum)*

*#the 2 here tells “apply” to apply to columns. A 1 would mean apply to rows.*

*table(sumlist)*

Paste the output here. Which outcome was the most common in your dice rolls?

> table(sumlist)

sumlist

2 4 5 6 7 8 9 11

1 2 5 3 4 1 2 2

14. Rewrite the replicate and apply commands from 11 and 13 to simulate rolling 5 dice 1000 times, and plot the totals of the five dice in a histogram.

sample(1:6, size = 5, replace = TRUE, prob = rep(1/6, 6))

dice2<-replicate(1000,sample(1:6, size = 2, replace = TRUE, prob = rep(1/6, 6))) #replicate does the sampling again and again

dice2

dice2 <- replicate(100, sample(1:6, size = 2, replace = TRUE, prob = rep(1/6, 6)))

sums <- numeric(100)

for (i in 1:ncol(dice2)) {

sums[i] <- sum(dice2[,i])

} sample(1:6, size = 5, replace = TRUE, prob = rep(1/6, 6))

dice2<-replicate(1000,sample(1:6, size = 2, replace = TRUE, prob = rep(1/6, 6))) #replicate does the sampling again and again

dice2

dice2 <- replicate(100, sample(1:6, size = 2, replace = TRUE, prob = rep(1/6, 6)))

sums <- numeric(100)

for (i in 1:ncol(dice2)) {

sums[i] <- sum(dice2[,i])

}

hist(dice2)

hist(dice2)

