Repeating Using Loops

## Repeating operations with a for() loop

If you want to iterate over a set of values, when the order of iteration is important, and perform the same operation on each, a for() loop will do the job. Avoid using for() loops unless the order of iteration is important: i.e. the calculation at each iteration depends on the results of previous iterations.

The basic structure of a for() loop is:

for(iterator in set of values){  
 do a thing  
}

For example:

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 1:10 bit creates a vector on the fly; you can iterate over any other vector as well.

We can use a for() loop nested within another for() loop to iterate over two things at once.

for(i in 1:5){  
 for(j in c('a', 'b', 'c', 'd', 'e')){  
 print(paste(i,j))  
 }  
}

## [1] "1 a"  
## [1] "1 b"  
## [1] "1 c"  
## [1] "1 d"  
## [1] "1 e"  
## [1] "2 a"  
## [1] "2 b"  
## [1] "2 c"  
## [1] "2 d"  
## [1] "2 e"  
## [1] "3 a"  
## [1] "3 b"  
## [1] "3 c"  
## [1] "3 d"  
## [1] "3 e"  
## [1] "4 a"  
## [1] "4 b"  
## [1] "4 c"  
## [1] "4 d"  
## [1] "4 e"  
## [1] "5 a"  
## [1] "5 b"  
## [1] "5 c"  
## [1] "5 d"  
## [1] "5 e"

Rather than printing the results, we could write the loop output to a new object.

output\_vector <- c()  
for(i in 1:5){  
 for(j in c('a', 'b', 'c', 'd', 'e')){  
 temp\_output <- paste(i, j)  
 output\_vector <- c(output\_vector, temp\_output)  
 }  
}  
output\_vector

## [1] "1 a" "1 b" "1 c" "1 d" "1 e" "2 a" "2 b" "2 c" "2 d" "2 e" "3 a"  
## [12] "3 b" "3 c" "3 d" "3 e" "4 a" "4 b" "4 c" "4 d" "4 e" "5 a" "5 b"  
## [23] "5 c" "5 d" "5 e"

This approach can be useful, but ‘growing your results’ (building the result object incrementally) is computationally inefficient, so avoid it when you are iterating through a lot of values.

## Tip: don’t grow your results

One of the biggest things that trips up novices and experienced R users alike, is building a results object (vector, list, matrix, data frame) as your for loop progresses. Computers are very bad at handling this, so your calculations can very quickly slow to a crawl. It’s much better to define an empty results object before hand of the appropriate dimensions. So if you know the end result will be stored in a matrix like above, create an empty matrix with 5 row and 5 columns, then at each iteration store the results in the appropriate location.

A better way is to define your (empty) output object before filling in the values. For this example, it looks more involved, but is still more efficient.

output\_matrix <- matrix(nrow=5, ncol=5)  
j\_vector <- c('a', 'b', 'c', 'd', 'e')  
for(i in 1:5){  
 for(j in 1:5){  
 temp\_j\_value <- j\_vector[j]  
 temp\_output <- paste(i, temp\_j\_value)  
 output\_matrix[i, j] <- temp\_output  
 }  
}  
output\_vector2 <- as.vector(output\_matrix)  
output\_vector2

## [1] "1 a" "2 a" "3 a" "4 a" "5 a" "1 b" "2 b" "3 b" "4 b" "5 b" "1 c"  
## [12] "2 c" "3 c" "4 c" "5 c" "1 d" "2 d" "3 d" "4 d" "5 d" "1 e" "2 e"  
## [23] "3 e" "4 e" "5 e"

## Repeating operations until a condition is met

Sometimes you will find yourself needing to repeat an operation as long as a certain condition is met. You can do this with a while() loop.

while(this condition is true){  
 do a thing  
 }

R will interpret a condition being met as “TRUE”. As an example, here’s a while loop that generates random numbers from a uniform distribution (the runif() function) between 0 and 1 until it gets one that’s less than 0.1.

z <- 1  
while(z > 0.1){  
 z <- runif(1)  
 cat(z, "\n")  
 }

## 0.5074782   
## 0.3067685   
## 0.4269077   
## 0.6931021   
## 0.08513597

while() loops will not always be appropriate. You have to be particularly careful that you don’t end up stuck in an infinite loop because your condition is always met and hence the while statement never terminates.

## Challenge 1

Write a script that loops through the mtcars data by cylinder number and prints out whether the mean mpg is lower or higher than 20 miles per gallon for each category of cyl (cylinders) in vehicles.

## Solution to Challenge 1

**Step 1**: We want to make sure we can extract all the unique values of the cylinder vector

unique(mtcars$cyl)

**Step 2**: We also need to loop over each of these cylinder categories and calculate the average mpg for each subset of data. We can do that as follows:

1. Loop over each of the unique values of ‘cyl’
2. For each value of cylinder, create a temporary variable storing that subset
3. Return the calculated mean mpg to the user by printing the output:

for( iCyl in unique(mtcars$cyl) ){  
 tmp <- mtcars[mtcars$cyl == iCyl, ]   
 cat(iCyl, mean(tmp$mpg, na.rm = TRUE), "\n")   
 rm(tmp)  
}