For\_loops

Martin Andrzejak

November 1, 2021

How for loops work and alternatives

In any programming language loops exist to make repetitive work easier. Each cycle of the loop is called an iteration and does exactly what the cycle before does but you can change a few things. In R, loops are usually coded like this: for(i in 1:x) Translated into English that basically means: for every i use 1-x. That means: i in the first iteration becomes 1, in the second 2 until the maximum number x is reached. Note that if you have only one line in the for loop (following example) you don't need to wrap the loop in these: {}, they are needed only if you have multiple lines in the loop. Let's follow with an example:

## loop 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

In this simple and always used example, i takes in each iteration the number from 1 - 10. Important to know is, each iteration of i is independent of the previous and will give you a different result. Let’s try another example which is a bit more complex:

for(i in 1:10)  
 print(2\*i)

## [1] 2  
## [1] 4  
## [1] 6  
## [1] 8  
## [1] 10  
## [1] 12  
## [1] 14  
## [1] 16  
## [1] 18  
## [1] 20

So in this example we calculate 2 \* 1:10, each iteration multiplies 2 by the value i (first iteration 2 by 1, second 2 by 2... last iteration 2 by 10) But why should we use for loops or anything else that does iterations when we could also simply do:

2\*1:10

## [1] 2 4 6 8 10 12 14 16 18 20

In this case, 2 also gets multiplied by 1:10. Well, something very useful in a for loop is that we can use an if statement (maybe the main reason why I use for loops most of the time). An if statement compares whatever you give to the computer to a condition. So something like this: If the result is > 10 print("I am > 10") else (if the condition is not met) print("I am < 10").

for(i in 1:10)  
{  
 if((2 \* i) > 10)  
 print(paste(2, "\*", i, "=", 2\*i, " I am > 10", sep = ""))  
 else  
 print(paste(2, "\*", i, "=", 2\*i, " I am < 10", sep = ""))  
}

## [1] "2\*1=2 I am < 10"  
## [1] "2\*2=4 I am < 10"  
## [1] "2\*3=6 I am < 10"  
## [1] "2\*4=8 I am < 10"  
## [1] "2\*5=10 I am < 10"  
## [1] "2\*6=12 I am > 10"  
## [1] "2\*7=14 I am > 10"  
## [1] "2\*8=16 I am > 10"  
## [1] "2\*9=18 I am > 10"  
## [1] "2\*10=20 I am > 10"

Still these are all easy examples, just to show you how for loops work. I will try to explain some alternatives; for example, there is another easier way for an if statement not including a for loop: ifelse()

ifelse(2\*1:10 > 10, "I am > 10", "I am < 10")

## [1] "I am < 10" "I am < 10" "I am < 10" "I am < 10" "I am < 10" "I am > 10"  
## [7] "I am > 10" "I am > 10" "I am > 10" "I am > 10"

This example is supposed to show you that there are other options in R and if you know one it is almost always better to avoid for loops, if it can be done in a reasonable amount of time and effort.

Now for loops usually get in handy when you have to do the same thing over and over again in a data frame or a matrix (See the Document: data classes R, for information on those). Now usually we don't want to print our results from a for loop but we want to store them, which is often a source for errors. Typical errors are:

1. One doesn't define the "place" to store the new result of each iteration and overwrites the result from the previous iteration which would look like this:

for(i in 1:10)

result <- 2\*i

In this case result takes the value of each iteration and overwrites the one from the previous. The final result will just be one number (in this case 20) instead of a vector of numbers.

1. You didn't create an empty vector/dataframe/list/matrix/array. While it is possible to create objects in a for loop:

for(i in 1:10)

result <- 2\**i*

It is not possible to create and fill something:

for(i in 1:10)

result[i] <- *2*1

What you need to do is to create the vector or anything beforehand like this:

result <- c()

for(i in 1:10)

result[i] <- 2\*i

1. Probably the biggest mistake is that you get the indexing of your storage wrong. For example if you want to fill a dataframe every i row you need to write it like this: dataframe[i,] as data frames are indext like this: dataframe[rownumber, columnnumber/name]
2. I have also seen people trying to store multiple things into one spot: result[i] <- output[1:5] In this case you try to store 5 values into one cell. You can picture it like this: You have a box where you can put exactly one coin in, but you try to put 5 in. It won’t work and you will lose all coins.

Of course, all those options and possibilities as well as the coding style is very much influenced on how I learned R, and there are a ton of better, faster and easier ways. There is no one right/wrong way, as long as it works.

Another alternative to a for loop is a repeat loop. In this case, it works very much like a for loop just that you define on what iteration the loop should break.

i <- 1  
repeat  
{  
 print(2\*i)  
   
 i <- 1+i  
   
 if(i > 10)  
 break  
}

## [1] 2  
## [1] 4  
## [1] 6  
## [1] 8  
## [1] 10  
## [1] 12  
## [1] 14  
## [1] 16  
## [1] 18  
## [1] 20

vapply(1:10, FUN.VALUE = 1, function(x){x \* 2})

## [1] 2 4 6 8 10 12 14 16 18 20