Loopy for Loops and Silly for Samples - Hands-on!

Dr B and Daphne Hansell (BMC '24)

2023-10-24

Outline

Hello R students! Your silly student coder has returned to make your assignments notably worse than if Dr. B had made them. This week you will learn:

- intro to loops: `for` and `while`.[1]
- conditional statement: `if`, `then`, `else`, and `ifelse`.
- a soft intro intro to sampling using mostly base R.
- as much dplyr as I can include to drive Levi crazy[2].

Part 1: Loops

Loops are a programming element that repeat a portion of code a set number of times until the desired process is complete.[3]

All programming languages have loops. Their syntax might differ, but they are always there, just like element types, data structures, etc.

`for` loops

First, we are going to be focusing on "for loops". A for loop applies a command to each value provided then stops. In other words, the `for` loop runs for a preset number of times.

Basic example:

```
for (i in 1:5) {
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

[1] R also has a loop called repeat that I only learned about when I googled "types of loops in R. I don't think you'll need to know about it.

[2] Ask Levi about his beef with `tidyverse` if you are unaware.

[3] See ref 1 at the end of this document.

- You don't actually have to use i for the value there but most people do. `i` is presumably short for "index", which makes sense. But you could just as well call it `banana`.

Ex.1) run the same command above replacing `i` with `banana`.

```
#type your code in here
```

So this loop printed each value for `i` (or `banana`). How would you write code that printed the values 6-10?

```
#type your code in here
```

It's often useful to define a placeholder variable before running your loop. For example, this loop calculates the mean of a given data set.

```
numbers <- c(4, 22, 6, 13, 19, 2, 11)
sum <- 0 #placeholder (or counter) variable
for(num in numbers) {
  sum <- sum + num
}
mean <- sum / length(numbers)
print(mean)
```

```
## [1] 11
```

You can also write loops with character vectors. Here's a character vector: `days<-c("Mon", "Tues", "Wednes", "Thurs", "Fri", "Satur", "Sun")`

Ex.2) Write a loop that adds day to each day of the week so that the output of `days` gives you "Monday", "Tuesday", etc, one in each line.

Hint: if you add the argument `sep = ""` to the `paste` function you don't get a space in between the things being pasted.

```
#First create the vector days as show above

#now write a for loop that adds "day" to each of the elements and print each one as you
↪  go
```

`while` loops

A `while` loop repeats code until a condition is met. It's really important to include the condition because the loop might continue forever if you don't.

Here's an example of a loop that prints x then multiplies it by two, as long as `x` is under `16`.

```
x <- 2
while (x < 16) {
  print(x)
  x <- x * 2
}
```

```
## [1] 2
## [1] 4
## [1] 8
```

Ex.3) Now write a while loop that prints the value, then doubles it, starting at 1 and stopping at 100.

```
#your code here
```

Just for fun try running that loop without with `while(x)`. This will run forever. Make sure you remember to add the condition! To quit this infinite loop, type ctl+C or ESC (Posit Cloud).

Say you have a number and you want to find the smallest divisor that isn't 1. You could use a while loop to do this.
Hint: remember the `%%` operator from week 1. It's called "modulo" operator.

```
#your code here
```

You can do a lot more with loops once you learn about functions. But alas, that's for next lab. For now that's probably enough for loops.

Recommended: This guide someone wrote on the internet is really good if you want to also read it. https://intro2r.com/loops.html

---

Part 2: Sampling

There is a basic function in R for taking samples very appropriately named sample. You need three arguments for it to work (there is a 4th also, probability). The first is the data set, the second is the number of samples, and the third is with or without replacement.

Ex.4) Let's start with some basic sampling. Imagine you have a coin, equal chances of heads and tails. You want to flip it 10 times and see how many heads and how many tails. First make a vector called `coin_toss`, with "heads" and "tails" as elements. Then, sample it 10 times, with replacement:

```
#your code here
coin_toss<-c("heads","tails")
sample(coin_toss, size=10, replace=TRUE)
```

```
##  [1] "heads" "tails" "tails" "tails" "heads" "heads" "tails" "heads" "tails"
## [10] "tails"
```

Now lets try and do this same thing with a for loop. Make an empty vector called `all_tosses` and use it to generate 10 coin tosses.

Hint:Because you are looping over the sample you are only sampling one each time

```
# first create an empty vector called all_tosses
all_tosses<-c()
#now run the loop

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

```
        #print(i)
        all_tosses[i]<-sample(coin_toss, size=1, replace=F)

}

print(all_tosses)
```

```
##  [1] "tails" "heads" "tails" "heads" "heads" "heads" "heads" "tails" "heads"
## [10] "tails"
```

Now I'm being sneaky. I challenge you to flip a coin but its weighted. It lands on heads 60% of the time. Use the same loop and add the probabilities to the sample function.

Hint: Use the `prob` argument in the `sample` function.

```
#first create an empty vector called all_tosses

#now run the loop
```

Now let's see what happens if we flip both coins even more times. Use your original loop with the fair coin and toss is 100 times. Then take the weighted coin, change the variable names to make them different (add the word weighted or rigged) and toss that 100 times. Then count how many heads and how many tails came from each coin. Make sure to make an empty vector for each loop. Hint: you can use the table function to get a count of heads and tails

```
#your code here
```

This is why you shouldn't play games with other peoples strange coins!

Ex.5) Read in the `human_genes.csv` file and name it `human_genes`. Play around with it using your strong exploratory data analysis skills. Get a good feel for the data set.

Overall goal of this exercise is to compare sample means at different sample sizes to the population mean.

Its big right? So let's take some samples.

NOTE: Your output will look different to the pdf because your samples will be different values.

5.1. First filter `human_genes` to only keep the size and name column.

```
#your code here
```

5.2. create a second data frame with summary statistics mean, median and SD of length. (what is interesting about this data??). Hint: use the `summarize` function.

```
#your code here
```

`5.3. Now let's take some samples. Take 3 samples of gene size with replacement on. Usen` = 10, 100, 1000. Calculate the mean of these samples, and save each mean as mean10, mean100, and mean1000.

```
#your code here
```

5.4. Compare these means to the population mean. Which is closest?

```
#your code here
```

Congrats, you did sampling!

Ex.6) Now let's combine this with the skills you learned in loops. You are going to take 100 samples of each of those sample sizes to compare

6.1 First you want to create 3 vectors to store each of your results. Them mean_values_10 etc.

```
#your code here
```

6.2 Now you write a for loop like you did before. Execept you can include all three of your new vectors in the same loop.

```
#your code here
```

You have now bootstrapped. Lets graph

Ex.7). We want to compare our samples to see which sample size got closest to our population mean. You are going to make a histogram of these means.

7.1 It's probably a good idea to start with making a data frame of our values. Take the vectors you just made and make one column called "mean" and another column with the corresponding sample size called "sample_size." You're going to have to make this a factor for the graphs to turn out right. You also need to nestle the functions `rep` inside your factor call, with the argument `each = 100`. Call your data frame `mean_data`.

```
#your code here
```

7.2 Lets make a basic histogram. x axis is mean, fill by sample_size.

```
#your code here
```

7.3 This is a bit hard to read. Let's add some detail. Inside geom_histogram, change position to "fill", and both alpha to 0.4 and binwidth to 30.

```
 #your code here
```

7.4 Much better! Ok add some labels on your axis and legend. perhaps even add a theme if you feel silly

```
#your code here
```

Are you tired of boring ggplot themes? Sick of theme_minimal? Run this code and go crazy devtools::install_github("https://github.com/MatthewBJane/ThemePark")

Data Camp activities

To strengthen the skills learned today:

- Intermediate R: Loops
- Sampling in R: Introduction to sampling
- Sampling in R: Bootstrap distributions

The end!

- Knit your document into a PDF using the button above this text editor and upload the PDF into Moodle.

References

[1] Kodable. What are loops? Link

[2] Intro2r. For loops. https://intro2r.com/loops.html