Loops Clinic

Dan Maxwell

May 16, 2017

Loops in R

There are three kinds of loop in R. The `for` loop is probably the most popular. Slide 1 provides a quick overview of this loop, its syntax and operation.

For loops

The for loop is a loop that is executed a specified number of times. In this example, a variable (i) steps through a vector of integers from 1 to 5, at which point the loop terminates.

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

Later in this clinic, we'll examine some code that demonstrates how a for loop works. But before we do that, let's take a quick look at R's other two loops. Slide 2 illustrates the syntax and logical operation of the `while` loop.

While loops

The while loop is often used for executing a set of commands or statements repeatedly until a specific condition is satisfied.

The structure of a while loop consists of a boolean condition and statements that are written inside while loop brackets, for which repetitive execution is to be carried out until the condition of interest is satisfied:

```
while (condition) {
    expression
    expression
    expression
}
```

It is important to note that the while loop will first check that the condition is satisfied prior to executing a first iteration of the commands.

And finally, Slide 3 presents the logical operation and basic syntax of the `repeat` loop.

Repeat loops

The repeat loop is an infinite loop that is often used in conjunction with a **break** statement that terminates the loop when a specified condition is satisfied. The basic structure of the repeat loop is:

```
repeat{
   expression
   expression
   expression
   if(condition) break
}
```

Sample Code

Now that a concise outline of R's three loop statements has been presented, we need to look at some actual code, to see how R loops work in the real world. To do that, we are going to load the arsenic data file and then step through it. Let's take a look at what we've got.

```
# Read the data file and then view its contents.

arsenic <- read.csv (file = "c:/informatics/arsenic.txt", header = TRUE, sep = "\t", as.is = TRUE)
head(arsenic)
```

```
##   age sex usedrink usecook arswater arsnails
## 1  44   F        E       E  0.00087    0.119
## 2  45   F        D       E  0.00021    0.118
## 3  44   M        E       E  0.00000    0.099
## 4  66   F        C       E  0.00115    0.118
## 5  37   M        B       E  0.00000    0.277
## 6  45   F        E       E  0.00000    0.358
```

The documentation that came with this dataset indicates that it contains informations from an arsenic study conducted in the state of New Hampshire. Each row provides measurements on one study participant. The age and sex of each individual is given as well as the arsenic concentration found in their water *(arswater)* and nails *(arsnails)*. The dataset also contains two categorical variables: *usedrink* and *usecook*. As such, these variables represent the percentage of well water each study participant used for drinking and cooking. Each category and its corresponding percentage is listed in the table below.

Category	Well Water Pct
A	$< 1/4$
B	$\sim 1/4$
C	$\sim 1/2$
D	$\sim 3/4$
E	$> 3/4$

Unfortunately, the dataset only provides the categories and not the well water percentages associated with each category. The practical consequence of this is that any reports we create with this dataset will only show the categories (A, B, C, D, E) and the viewer has to mentally translate those values into the corresponding percentages. Thus what we need to do is add these percentages directly to the dataframe, thereby making them available in our reports.

To do this, we'll need to loop through the rows in the dataframe, assigning a value to a new variable, based on the category (A, B, C, D, and E) in either usedrink or usecook. The code below shows how to do this for the *usedrink* variable and the same logic would apply for *usecook*. An example for each looping mechanism is provided.

```
# R supports for, while, and repeat loops.  The for loop simply executes
# a block of code a set number of times, initializing an idx and then incrementing
# it each iteration of the loop.  This is probably the most popular loop in R.

for(idx in 1:nrow(arsenic)){
  arsenic$usedrinkpct[idx] <- "NA"       # Assign a default to the variable.

  if (arsenic$usedrink[idx] == "A") {arsenic$usedrinkpct[idx] <- "< 1/4"}
```

```
  if (arsenic$usedrink[idx] == "B") {arsenic$usedrinkpct[idx] <- "~ 1/4"}
  if (arsenic$usedrink[idx] == "C") {arsenic$usedrinkpct[idx] <- "~ 1/2"}
  if (arsenic$usedrink[idx] == "D") {arsenic$usedrinkpct[idx] <- "~ 3/4"}
  if (arsenic$usedrink[idx] == "E") {arsenic$usedrinkpct[idx] <- "> 3/4"}
}

# The while loop checks that a logical condition is true before entering the
# loop.  In the example below, we would not even enter the while loop if the
# idx variable was greater than the number of rows (nrow) in the arsenic data
# frame. The while condition is checked at each iteration of the loop.

idx <- 1

while (idx <= nrow(arsenic)) {
  arsenic$usedrinkpct[idx] <- "NA"       # Assign a default to the variable.

  if (arsenic$usedrink[idx] == "A") {arsenic$usedrinkpct[idx] <- "< 1/4"}
  if (arsenic$usedrink[idx] == "B") {arsenic$usedrinkpct[idx] <- "~ 1/4"}
  # ... the other if statements.

  idx <- idx + 1
}

# The repeat loop, unlike the previous two examples, does not have a logical
# check at the beginning of its execution.  That is, it immediate begins to
# loop through and execute a block of code until a break statement is
# encountered.  Absent such a statement, the repeat loop continues forever,
# or until someone manually forces it to stop.  Thus one MUST provide a way
# to break out of the loop at some point.

idx <- 1

repeat {
  arsenic$usedrinkpct[idx] <- "NA"       # Assign a default to the variable.

  if (arsenic$usedrink[idx] == "A") {arsenic$usedrinkpct[idx] <- "< 1/4"}
  if (arsenic$usedrink[idx] == "B") {arsenic$usedrinkpct[idx] <- "~ 1/4"}
  # ... the other if statements.

  if (idx >= nrow(arsenic)) {              # Break out of loop at last row.
    break
  }

  idx <- idx + 1
}
```