Intro R conditional executions and loops

Roberto Molowny-Horas

April 24-25, 2023

Conditional executions

Similar to other programming languages, R has commands to perform conditional executions. That allows us to select subsets of the original data set that satisfy a set of conditions.

```
a <- 1:3

if ((a[1]>0 & a[1]<=2)) b <- 4 else b <- 3      # if larger than 0 and smaller than, or equal to, 2.

if ((a[1]==0 | a[1]==2)) b <- 4 else b <- 3     # if equal to 0 or equal to 2.

if (a[1]!=0 ) b <- 4 else b <- 3                # if not equal to 0.

# Vectorization.
a <- rnorm(1000)
b <- (a>1.96)*1 + (-1.96<=a & a<=1.96)*0 + (-1)*(a<(-1.96))
hist(b)
```

Exercise 1. Create a small data frame with two columns. Column #1 will contain numbers from 1 to 20 in increasing order. Column #2 will contain text stating that the corresponding number in column #1 is even or odd. Use R function **round** *to check for evenness or oddness*.

Loops

In R, as in other computer languages, a loop is a way of repeatedly carrying out tasks or computations over the elements of a data set or a data structure. In R many of those loops can be vectorized, which saves much computing time and allows a better code readability. In other cases, although vectorization may not be possible, R provides many loop functions, like e.g. **for**, **sapply**, **lapply**, **tapply**, **vapply**, **replicate**. Conditional loops, i.e. loops which stop only after one or several conditions are met, can also be implemented with **while** or **repeat**.

Below we will briefly explore the usage of **for**, which is the most explicit and, probably, the most intuitive of all looping functions. If time or code readability are not your concerns, **for** is probably good enough in most cases. However, we recommend that you have a look at the help pages of the other functions, as well as the many teaching resources on the internet, for more information.

Without further ado, let’s show an example of the use of **for**.

```r
a <- runif(10)
b1 <- vector("numeric", length(10))
for (i in 1:10) {
  if (a[i] > .5) {
    b1[i] <- 1
  } else {
    b1[i] <- 0
  }
}

# This can also be done in a vectorized way.
b2 <- ifelse(a>.5,1,0)          # The simplest way.
b3 <- as.numeric(a>.5)          # The more compact way.
```

If you are interested in knowing how it would work with **sapply**, this is how (although in this particular case, the vectorized way should be our preferred choice).

```r
b4 <- sapply(1:10, function(i) (a[i]>.5)*1)
```

Finally, the **apply** function may be useful when trying to calculate along the indices of a matrix or data.frame.

```r
a <- matrix(runif(1000*100), 1000, 100)
ma <- apply(a, 1, sum)    # Keep each row fixed and sum over the corresponding 100 co
lumns.
qa <- apply(a, 2, function(x) quantile(x, probs = .5))
```

*Exercise. Do “data(iris)” and use function **sapply** to calculate the average value of the first 4 columns.