Chapter 7

Programs

After an overview of R's capabilities and functions, this chapter will be devoted to some basic programming concepts.

7.1 Conditionals

One of the most basic programing concepts involves evaluating conditional statements and then performing some action based on the evaluation. Let's write a very simple conditional using the `if()` control-flow construct. The `if()` construct is simply a control statement that takes a conditional statement as its argument and then depending on the evaluation initiates some other function. Let's ask R if `1 + 1 = 2` and if so to print something to the screen with the `print()` function:

```
> if(1 + 1 == 2) print("Definitely!")
[1] "Definitely."
>
```

You can see that in this case the `if()` statement evaluated to `TRUE` and it activated the `print()` function. If the `if()` statement were `FALSE` nothing would happen. Try it.

```
> if(1 + 1 == 3) print("Definitely.")
>
```

If we wanted to R to also do something if the conditional is `FALSE` we would have to add the `else` statement.

```
> if(1 + 1 == 3) print("Definitely.") else print("Wrong.")
[1] "Wrong."
>
```

This can be pretty useful especially since we can ask R to perform much more complicated tasks. Before we try this I want to introduce you to some of R's random number generating functions.[1]

Function	Description
rnorm()	random variates for the normal distribution (arguments: n, mean, sd)
runif()	random variates for the uniform distribution (arguments: n, min, max)
rbinom()	random variates for the binomial distribution (arguments: n, size, prob)
rweibull()	random variates for the Weibull distribution (arguments: n, shape, scale)

Let's use these random number generators within an more complicated if() statement. To combine multiple tasks R should perform we have to use curly braces { }. Copy the following code and tell me what this program is doing.

```
 Test <- runif(n = 1, min = 0, max = 2)
 if(Test < 1) {
    X <- rnorm(n = 1000, mean = Test, sd = 1)
    ggplot() +
      geom_density(aes(X), fill = "red") +
      labs(title = paste("Mean = Test =", Test),
        x = "1000 Random Normal Deviates, sd = 1",
        y = "Density")
  } else {
     Y <- rweibull(n = 1000, shape = Test, scale = 1)
     ggplot() +
      geom_density(aes(Y), fill = "green") +
      labs(title = paste("Shape = Test =", Test),
        x = "1000 Random Weibull Deviates, scale = 1",
        y = "Density")
  }
>
```

7.1.1 The ifelse() Function

Instead of using the if() and else control statements, you can just use the ifelse() function. The function takes the arguments test, yes and no.

```
> Z <- 1:10
> Z
[1]  1  2  3  4  5  6  7  8  9 10
> Z.new <- ifelse(test = Z > 2 & Z < 8, yes = "Yes", no = "No
   ")
> Z.new
[1] "No" "No" "Yes" "Yes" "Yes" "Yes" "Yes" "No" "No" "No"
```

[1] For a more complete list type ?Distributions.

This function comes in very handy to do data manipulation. Suppose we wanted to create a dummy variable in our Students dataset such that FirstYears = 1, else 0. As always we can drop the argument names and write something like this:

```
> library(foreign)
> Students <- read.dta("http://www.peterhaschke.com/Teaching/
   R-Course/Students.dta")
> names(Students)
[1] "Name" "Year"

> Students$FirstYears <- ifelse(Students$Year == 1, 1, 0)
> names(Students)
[1] "Name" "Year" "FirstYears"

> ifelse(Students$FirstYears == 1, print(as.character(
   Students$Name)), print("NA"))
 [1] "NA"                 "NA"
 [3] "NA"                 "Jonathan Bennett"
 [5] "Peter Bils"         "NA"
 [7] "Hun Chung"          "NA"
 [9] "NA"                 "NA"
[11] "David Gelman"       "NA"
[13] "NA"                 "YeonKyung Jeong"
[15] "Doug Johnson"       "Gleason Judd"
[17] "NA"                 "NA"
[19] "NA"                 "NA"
[21] "NA"                 "NA"
[23] "NA"                 "NA"
[25] "NA"                 "NA"
[27] "Justin Nicholson"   "NA"
[29] "NA"                 "Barbara Piotrowska"
[31] "NA"                 "NA"
[33] "NA"                 "NA"
[35] "Jeheung Ryu"        "NA"
[37] "NA"                 "Bradley Smith"
[39] "NA"                 "NA"
[41] "NA"                 "NA"
[43] "NA"                 "Matthew Sweeten"
[45] "NA"                 "NA"
[47] "Jie Wen"
```

7.1.2 Nested Control Flow Statements

If-statements can easily be nested. This can become quite ugly looking and you may easily lose track of your conditions.

```r
> r <- runif(n = 1, min = 0, max = 1)
> if(r < 0.2){
  cat("r is", r, "which is smaller than 0.2")
  } else {
    if(0.2 < r & r < 0.5 ) {
      cat("r is", r, "which is between 0.2 and 0.5")
    } else {
    if(r > 0.5 & r < 0.9) {
      cat("r is", r, "which is greater than 0.5")
    } else {
      cat("r is", r, "which is greater than 0.9")
    }
  }
}
```

Things tend to be a bit cleaner with the build in `ifelse()` function.[2]

```r
ifelse(r < 0.2,
  paste("r is", r, "which is smaller than 0.2"),
  ifelse(0.2 < r & r < 0.5,
    paste("r is", r, "which is between 0.2 and 0.5"),
    ifelse(r > 0.5 & r < 0.9,
      paste("r is", r, "which is greater than 0.5"),
      paste("r is", r, "which is greater than 0.9")
    )
  )
)
```

[2] Notice the use of the `paste()` function. We have to use `paste()` with the `ifelse()` function because `ifelse()` does not play nice with `cat()`. Ultimately, `paste()` does the same thing but it implicitly creates an object (i.e. a character vector) whereas `cat()` does not. Since you cannot assign the output of `cat()`, `ifelse()` which is a function manipulating and returning objects, breaks. The `if()` statement is not a function but a control flow operator and doesn't care if an object is created after the logical evaluation or not.