Advanced R Programming

Xuemao Zhang
Department of Mathematics
East Stroudsburg University

June 24, 2019

Outline

- R condition statements
- R loops
- R functions
- Data manipulations

R condition statement: `if`

- Condition statemants allow you to specify the execution of your code. They are extremely useful if you want to run a piece a code if a certain condition is met. The syntax of `if` statement is

```
if (test_expression) {statements;}
```

```r
x = c(8, -3, 2, -6);
#any() checks if any of the elements of a vector are TRUE
if(any(x < 0)) print("x contains negative numbers");
```

```
## [1] "x contains negative numbers"
```

```r
if(any(x < 0))
  {print("x contains negative numbers");
  print(x[which(x<0)]);
#which() function returns the positions of the elements
}
```

```
## [1] "x contains negative numbers"
## [1] -3 -6
```

R condition statement: `if...else`

- The conditional `if ... else` statement is used to test an expression similar to the `if` statement. However, if the `test_expression` is FALSE, the `else` part of the function will be evaluated. The syntax is

```
if (test_expression) {
        statement 1;
} else {
        statement 2;
}
```

R condition statement: `if...else`

```r
x = c(8, -3, 2, -6);
if(any(x < 0))
  {print("x contains negative numbers");
  print(x[which(x<0)]);
} else
{print("x contains all positive numbers")}
```

```
## [1] "x contains negative numbers"
## [1] -3 -6
```

```r
x = c(8, 3, 2, 6);
if(any(x < 0))
  {print("x contains negative numbers");
  print(x[which(x<0)]);
} else
{print("x contains all positive numbers")}
```

```
## [1] "x contains all positive numbers"
```

R condition statement: `if...else`

- We can also nest as many `if...else` statements as required (or desired).

```r
k = 21;
if(k > 20){
  print("The number is greater than 20");
} else if (k < 20){
  print("The number is less than 20");
} else {
  print ("The number is equal to 20");
}
```

```
## [1] "The number is greater than 20"
```

R condition statement: `if...else`

- `ifelse()` function is a shorthand function to the traditional `if...else` statement. The syntax is

```
ifelse(test_expression, x, y)
```

- The `test_expression` must be a logical vector (or an object that can be coerced to logical).
- This returned vector has element from `x` if `test_expression` is `TRUE` or from `y` if `test_expression` is `FALSE`.

```
x = c(9,4,0,-4,-9);
sqrt(x);   #it gives warning
```

```
## Warning in sqrt(x): NaNs produced
```

```
## [1]   3   2   0 NaN NaN
```

```
sqrt(ifelse(x >= 0, x, NA))  # no warning
```

```
## [1]  3  2  0 NA NA
```

R loops: `for` loop

- Loops are used in programming to repeat a specific block of code.
- Only the `for` loop will be used for all examples in this summer institute.
- The `for` loop is used to execute repetitive code statements for a particular number of times. The syntax is

```
for (val in sequence)
{
statements;
}
```

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

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

R loops: `while` loop

- We skip the other loops in the following. Read them after the summer institute.

- `While` loops begin by testing a condition. If it is `TRUE`, then they execute the statement. Once the statement is executed, the condition is tested again, and so forth, until the condition is `FALSE`, after which the loop exits.

- The syntax is

```
while(condition)
{expressions;}
```

```
i=0;
while (i < 5)
  {
    print(paste("i is", i));
    i=i+1;
  }
```

```
## [1] "i is 0"
## [1] "i is 1"
## [1] "i is 2"
## [1] "i is 3"
```

R loops: repeat loop

- A repeat loop is used to iterate over a block of code multiple number of times. There is test expression in a repeat loop to end or exit the loop. Rather, we must put a condition statement explicitly inside the body of the loop and use the break function to exit the loop. Failing to do so will result into an infinite loop.
- The syntax is

```
counter = 1;
repeat {
        statements;
        if(test_expression){
                break;
        }
        counter = counter + 1;
}
```

R loops: repeat loop

```r
x = 1;
repeat {
print(x);
x = x+1;
if (x == 6){
break;
}
}
```

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

R loops: `break` and `next`

- A `break` statement is used inside a loop (`repeat`, `for`, `while`) to stop the iterations and flow the control outside of the loop. It is used to exit a loop immediately if the `test_expression` is `TRUE`.

```
if (test_expression) {break;}
```

```r
for (i in 1:100) {
if (i == 4){
break;
}
print(i);
}
```

```
## [1] 1
## [1] 2
## [1] 3
```

R loops: `break` and `next`

- A `next` statement is useful when we want to skip the current iteration of a loop without terminating it. On encountering `next`, the R parser skips further evaluation and starts **next iteration** of the loop. The syntax is

```
if (test_condition) {next;}
```

```
for (i in 1:5) {
if (i == 3){
next;
}
print(i);
}
```

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

R functions

- A function is a set of statements organized together to perform a specific task.
- Functions are used to logically break our code into simpler parts which become easy to maintain and understand.
- R has a large number of in-built functions.

```
x = 1:50;
x;
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
## [24] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
## [47] 47 48 49 50
```

```
sum(x);  #Find the sum of the numbers
```

```
## [1] 1275
```

```
mean(x); #Find the average of the numbers
```

```
## [1] 25.5
```

R functions

- One of the great strengths of R is the user's ability to add functions.user can create their own functions. In fact, many of the functions in R are actually functions of functions. The structure of a function is

```
myfunction <- function(arg1, arg2, ... )
{
statements;
return(object);
}
```

- Function Name (myfunction): This is the actual name of the function. It is stored in R environment as an object with this name.
- Arguments (arg1, arg2, . . .): An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments are optional; that is, a function may contain no arguments. Also arguments can have default values.
- Function Body (statements): The function body contains a collection of statements that defines what the function does.
- Return Value (return(object)): The return value of a function is the last expression in the function body to be evaluated.

R functions

```r
# sum of the first n integers
summ<-function(n)
{
a=0;
for(i in 1:n)
{a=a+i;}
return(a);
}
summ(10)
```

```
## [1] 55
```

R functions

```r
# power of a vector or matrix
getPower<-function(x ,power)
{
   out<-x^power;
   return(out);
}
getPower(x=3,power=2);
```

```
## [1] 9
```

```r
getPower(x=c(1,2,3),power=2);
```

```
## [1] 1 4 9
```

Data manipulation: Sorting

- We have discussed several data manipulation methods.
- To sort a data frame in R, use the `order( )` function. By default, sorting is ASCENDING.

```r
setwd("F:/DataCamp/Day1");
quiz = read.csv("F:/DataCamp/data/quiz2.csv",header=TRUE, sep=",");
attach(quiz); #database is searched by R when evaluating a
#variable, no need to use
#detach it when it is not in use
quiz1 = quiz[order(Q1),];  #sort by Q1
head(quiz1);
```

```
##    ID Q1 Q2 Q3   Q4 Q5 Q6
## 4   4  6  5  9  8.0  5 NA
## 12 12  6  8  5  8.0 10  8
## 29 29  6  9  4  5.0  9  0
## 38 38  6  9 10  9.5  9  8
## 1   1  8  9 10  9.5 10  8
## 2   2  8  8  8 10.0  9  8
```

Data manipulation: Sorting

```
quiz2 = quiz[order(Q1,Q2),]; # sort by Q1 and Q2
head(quiz2);
```

```
##    ID Q1 Q2 Q3   Q4 Q5 Q6
## 4   4  6  5  9  8.0  5 NA
## 12 12  6  8  5  8.0 10  8
## 29 29  6  9  4  5.0  9  0
## 38 38  6  9 10  9.5  9  8
## 36 36  8  7  5  8.0  5  0
## 2   2  8  8  8 10.0  9  8
```

Data manipulation: Sorting

```r
quiz3 = quiz[order(Q1,-Q2),];
# sort by Q1 (ascending) and Q2 (descending)
head(quiz3);
```

```
##    ID Q1 Q2 Q3  Q4 Q5 Q6
## 29 29  6  9  4 5.0  9  0
## 38 38  6  9 10 9.5  9  8
## 12 12  6  8  5 8.0 10  8
## 4   4  6  5  9 8.0  5 NA
## 10 10  8 10 10 4.0 10  7
## 1   1  8  9 10 9.5 10  8
```

```r
detach(quiz);
```

Data manipulation: Merging

- To merge two data frames horizontally(adding columns), use the `merge()` function. You join two data frames by one or more common key variables.

```
exam = read.csv("F:/DataCamp/data/exam.csv",header=TRUE, sep=",");
grade=merge(quiz,exam,by="ID");
head(grade);
```

```
##    ID Q1 Q2 Q3   Q4 Q5 Q6   T1   T2   T3
## 1  1  8  9 10  9.5 10  8 21.5 16.5 23.5
## 2  2  8  8  8 10.0  9  8 21.0 16.0 20.0
## 3  3 10  7 10 10.0 10  8 30.5 31.0 30.0
## 4  4  6  5  9  8.0  5 NA 23.0 18.5 25.5
## 5  5 10  6  8  6.0 NA NA 25.5 13.5   NA
## 6  6 NA  9 10 10.0 10 NA 27.5 27.0 35.5
```

Data manipulation: Subsetting

- The function complete.cases() checks which observations/rows have no missing values.

- The subset( ) function is the easiest way to select variables and observations.

```
grade0 = grade[complete.cases(grade), ];#Keep the complete rows only
#Or use the suset() function
grade0=subset(grade, complete.cases(grade) == T);
str(grade0); dim(grade0);
```

```
## 'data.frame':    32 obs. of  10 variables:
##  $ ID: int  1 2 3 7 8 9 10 12 13 14 ...
##  $ Q1: int  8 8 10 10 10 10 8 6 10 10 ...
##  $ Q2: int  9 8 7 5 10 10 10 8 10 10 ...
##  $ Q3: int  10 8 10 9 10 6 10 5 8 10 ...
##  $ Q4: num  9.5 10 10 8 9 7 4 8 10 10 ...
##  $ Q5: int  10 9 10 10 10 8 10 10 10 9 ...
##  $ Q6: int  8 8 8 7 7 10 7 8 9 8 ...
##  $ T1: num  21.5 21 30.5 25.5 27.5 26 24.5 24 26.5 26 ...
##  $ T2: num  16.5 16 31 16 18.5 20.5 27.5 19 25.5 27 ...
##  $ T3: num  23.5 20 30 21.5 36 33 32.5 31.5 26.5 36.5 ...
```

Data manipulation: Subsetting

- The function `is.na()` indicates which elements are missing.
- For example, we replace all missing values in the data set “grade” by 0.

```
grade[is.na(grade)] = 0;
head(grade); dim(grade);
```

```
##   ID Q1 Q2 Q3   Q4 Q5 Q6   T1   T2   T3
## 1  1  8  9 10  9.5 10  8 21.5 16.5 23.5
## 2  2  8  8  8 10.0  9  8 21.0 16.0 20.0
## 3  3 10  7 10 10.0 10  8 30.5 31.0 30.0
## 4  4  6  5  9  8.0  5  0 23.0 18.5 25.5
## 5  5 10  6  8  6.0  0  0 25.5 13.5  0.0
## 6  6  0  9 10 10.0 10  0 27.5 27.0 35.5

## [1] 40 10
```

Data manipulation: Subsetting

- Selecting (keeping) variables.

```r
keep=c("Q1","Q2","T1","T2"); #Keep these 4 variables only
grade1= subset(grade, select=keep);
str(grade1);
```

```
## 'data.frame':    40 obs. of  4 variables:
##  $ Q1: num  8 8 10 6 10 0 10 10 10 8 ...
##  $ Q2: num  9 8 7 5 6 9 5 10 10 10 ...
##  $ T1: num  21.5 21 30.5 23 25.5 27.5 25.5 27.5 26 24.5 ...
##  $ T2: num  16.5 16 31 18.5 13.5 27 16 18.5 20.5 27.5 ...
```

Data manipulation: Subsetting

- Excluding (dropping) variables.

```r
grade2= subset(grade, select=-c(Q4,Q5,Q6,T3));
 #drop these 4 variables
str(grade2);
```

```
## 'data.frame':    40 obs. of  6 variables:
##  $ ID: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Q1: num  8 8 10 6 10 0 10 10 10 8 ...
##  $ Q2: num  9 8 7 5 6 9 5 10 10 10 ...
##  $ Q3: num  10 8 10 9 8 10 9 10 6 10 ...
##  $ T1: num  21.5 21 30.5 23 25.5 27.5 25.5 27.5 26 24.5 ...
##  $ T2: num  16.5 16 31 18.5 13.5 27 16 18.5 20.5 27.5 ...
```

Data manipulation: Subsetting

- Selecting observations.

```
grade3 = grade[1:5,];  # first 5 observations
grade3
```

```
##   ID Q1 Q2 Q3   Q4 Q5 Q6   T1   T2   T3
## 1  1  8  9 10  9.5 10  8 21.5 16.5 23.5
## 2  2  8  8  8 10.0  9  8 21.0 16.0 20.0
## 3  3 10  7 10 10.0 10  8 30.5 31.0 30.0
## 4  4  6  5  9  8.0  5  0 23.0 18.5 25.5
## 5  5 10  6  8  6.0  0  0 25.5 13.5  0.0
```

```
grade4 = subset(grade, grade$ID >= 31);
# based on variable values
grade4
```

```
##    ID Q1 Q2 Q3   Q4 Q5 Q6   T1   T2   T3
## 31 31 10 10 10  7.0  6  0 31.5 30.5 34.0
## 32 32 10 10 10 10.0  9  9 24.0 31.5 35.5
## 33 33 10  8  8  6.0  9  7 29.0 25.0 25.0
## 34 34 10 10  6  8.0  8  9 28.0 16.0 30.0
## 35 35 10 10  7  8.0 10 10 32.0 21.0 31.5
```

Data manipulation: Adding variables

- We can add new variables to a dataframe.

```r
grade[ , c("Total","Final")] = NA;
# add two more variables with values missing
str(grade);
```

```
## 'data.frame':    40 obs. of  12 variables:
##  $ ID   : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Q1   : num  8 8 10 6 10 0 10 10 10 8 ...
##  $ Q2   : num  9 8 7 5 6 9 5 10 10 10 ...
##  $ Q3   : num  10 8 10 9 8 10 9 10 6 10 ...
##  $ Q4   : num  9.5 10 10 8 6 10 8 9 7 4 ...
##  $ Q5   : num  10 9 10 5 0 10 10 10 8 10 ...
##  $ Q6   : num  8 8 8 0 0 0 7 7 10 7 ...
##  $ T1   : num  21.5 21 30.5 23 25.5 27.5 25.5 27.5 26 24.5 ...
##  $ T2   : num  16.5 16 31 18.5 13.5 27 16 18.5 20.5 27.5 ...
##  $ T3   : num  23.5 20 30 25.5 0 35.5 21.5 36 33 32.5 ...
##  $ Total: logi  NA NA NA NA NA NA ...
##  $ Final: logi  NA NA NA NA NA NA ...
```

Data manipulation: Adding variables

```r
grade$Letter = NA;
# add one more variable with values missing
str(grade);
```

```
## 'data.frame':    40 obs. of  13 variables:
##  $ ID    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Q1    : num  8 8 10 6 10 0 10 10 10 8 ...
##  $ Q2    : num  9 8 7 5 6 9 5 10 10 10 ...
##  $ Q3    : num  10 8 10 9 8 10 9 10 6 10 ...
##  $ Q4    : num  9.5 10 10 8 6 10 8 9 7 4 ...
##  $ Q5    : num  10 9 10 5 0 10 10 10 8 10 ...
##  $ Q6    : num  8 8 8 0 0 0 7 7 10 7 ...
##  $ T1    : num  21.5 21 30.5 23 25.5 27.5 25.5 27.5 26 24.5 ...
##  $ T2    : num  16.5 16 31 18.5 13.5 27 16 18.5 20.5 27.5 ...
##  $ T3    : num  23.5 20 30 25.5 0 35.5 21.5 36 33 32.5 ...
##  $ Total : logi  NA NA NA NA NA NA ...
##  $ Final : logi  NA NA NA NA NA NA ...
##  $ Letter: logi  NA NA NA NA NA NA ...
```

Data manipulation: Data type conversion

- Use `is.Type()` to test for data Type.
- Use `as.Type` to explicitly convert it.

```
is.numeric(),    as.numeric()
is.character(),  as.character()
is.vector(),     as.vector()
is.matrix(),     as.matrix()
is.data.frame(), as.data.frame()
```

```r
mtcars1=mtcars;
mtcars1$cyl=as.factor(mtcars1$cyl);
str(mtcars1);
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : Factor w/ 3 levels "4","6","8": 2 2 1 2 3 2 3 1 1 2 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ..
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
```

Data manipulation: Aggregating Data

- aggregate() function splits the data into subsets, computes summary statistics for each, and returns the result in a convenient form.
- The by variables must be in a list (even if there is only one).

```
# for numeric variables only
# aggregate data frame mtcars by cyl
# returning means

attach(mtcars);
aggdata =aggregate(mtcars, by=list(cyl),FUN=mean, na.rm=TRUE);
print(aggdata);
```

```
##   Group.1      mpg cyl     disp        hp     drat       wt     q
## 1       4 26.66364   4 105.1364  82.63636 4.070909 2.285727 19.13
## 2       6 19.74286   6 183.3143 122.28571 3.585714 3.117143 17.97
## 3       8 15.10000   8 353.1000 209.21429 3.229286 3.999214 16.77
##          vs        am     gear     carb
## 1 0.9090909 0.7272727 4.090909 1.545455
## 2 0.5714286 0.4285714 3.857143 3.428571
## 3 0.0000000 0.1428571 3.285714 3.500000
```

Questions?

?