CHAPTER 21

R Loops

Loops are very important in R since we use various loops to manipulate data, get a subset, merge different data sets, or redirect the flows of our programs. The most used loops are the **for()** loop and the **while()** loop. Let's look at the simplest loop of printing three values on the screen.

```
> for(i in 1:3) print(i)
[1] 1
[1] 2
[1] 3
```

The **i in 1:3** means that the variable i will take one value at a time from 1 to 3. The above one-line codes are equivalent to the following two lines.

```
> x<-1:10
> for(i in x)print(i)
```

Below is a similar example but with a character vector instead. A vector is a column variable with n-by-1 dimensions.

```
> tickers<-c("DELL","IBM","C","MSFT")
> for(ticker in tickers) print(ticker)
[1] "DELL"
[1] "IBM"
[1] "C"
[1] "MSFT"
```

21.1. For Loop

In the following program, the variable of i (a scalar variable) will take one value at a time from 1 to n, where *n* is the number of the total observations of x, which itself is a vector variable.

```
> x<-5:15
> n<-length(x) # get the number of observations for x
> for(i in 1:n) print(x[i])# print each item from x
```

For a multiple-line program, we should use a pair of curly braces, { and }, to circle those command lines.

```
for(i in 1:10){
 #
 # add your codes here
 #
 print(i)
}
```

There are some predefined data sets in R. For example, a character data set called **LETTERS** contains twenty-six capital letters while the data set called **letters**, its counterpart, contains twenty-six lowercase letters.

```
> letters
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
> typeof(letters)
                [1] "character"
> length(letters)
 [1] 26
```

The following codes show the variable called **letters**.

```
> for(letter in letters) print(letter)
```

Using the **cat()** function instead of the **print()** function. We will print all twenty-six letters one after another.

```
> for(letter in letters) cat(letter)
abcdefghijklmnopqrstuvwxyz>
```

If we intend to print one letter every line, we can add a new-line character ("**\n**").

```
> for(letter in letters) cat(letter, "\n")
```

The following program prints each ticker in a character vector called **tickers**.

```
> tickers<-c("IBM","DELL","MSFT")
> for(ticker in tickers)
> print(ticker)
```

21.2. Using Modulus Function to Shape the Output Format

Note that the function **%%** is the modulus function. It gives us the remainder **n %% m**. For instance, **11%% 10** will be 1, and **23 %% 10** will be 3. If we want to add a new line of "**\n**" every five letters, we can use the following codes.

```
n<-length(letters)
for(i in 1:n){
      cat(letters[i])
      if(i%%5==0) cat("\n")
}
# output shown on the right
```

```
abcde
fghij
klmno
pqrst
uvwxy
z>
```

21.3. Double Loops

For double loops, usually we use **[i,j]**, referring to those two loops. Again, properly indented codes are more readable.

```
n1<- 10
n2<- 50
for(i in 1:n1) {
   for(j in 1:n2){
        #
        # your codes here
        #
   }
}
```

For example, we intend to add 5 to the major diagonal variables **y[i, i]**, where **i=1, 2,..., 5** of a square matrix. The major diagonal line of a square matrix is from NW (northwest) to SE (southeast). First we generate a square matrix.

```
> x<-1:49 # x is a vector
> y<-matrix(x,7,7,byrow=T) # y is 7 by 7 matrix
> y
[,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,] 1 2 3 4 5 6 7
[2,] 8 9 10 11 12 13 14
[3,] 15 16 17 18 19 20 21
[4,] 22 23 24 25 26 27 28
[5,] 29 30 31 32 33 34 35
[6,] 36 37 38 39 40 41 42
[7,] 43 44 45 46 47 48 49
```

Below are the codes to add a value of 5 to the data items on the major diagonal line.

```
n1<-nrow(y)
n2<-ncol(y)
for(i in 1:n1) {
  for(j in 1:n2){
  if(i==j) y[i,j]=y[i,j]+5
  }
}
```

To double check that we have indeed added 5 to those values, type **y**.

```
> y
[,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,] 6 2 3 4 5 6 7
[2,] 8 14 10 11 12 13 14
[3,] 15 16 22 18 19 20 21
[4,] 22 23 24 30 26 27 28
[5,] 29 30 31 32 38 34 35
[6,] 36 37 38 39 40 46 42
[7,] 43 44 45 46 47 48 54
```

21.4. While Loop

Below is an example of a while loop:

```
i <- 0
while(i<15) {
      i<- i+2
      print(i)
}
```

Why are the following codes problematic?

```
i <- 0 # wrong codes
while(i<15) {
  print(i+2)
}
```

21.5. How to Stop (Cancel) an Execution

To stop a current computation, we can click **Misc** on the menu bar and choose **Stop current computation** (see below).

21.6. Stopping After Detecting an Error

Setting breaks would make our debugging efforts more efficient.

```
# add a break when an error happens
dd<-function(n){
      if (is.numeric(n)==FALSE) stop("Input should be numeric!")
      return(2*n)
}
```

After activating the function, we can test it by using the following two commands: one is for an integer input, and the other is for a character input.

```
> dd(2)
[1] 4
> dd("live")
Error in double("live") : Input should be numeric!
```

21.7. Length of a Vector vs. Dimension of a Matrix

A *vector* is defined as "a column variable," and we use the **length()** function to find out its number of observations.

```
> x<-c(1,2,4.5,7,9)
> x
[1] 1.0 2.0 4.5 7.0 9.0
> length(x)
[1] 5
```

A matrix is a two-dimensional data set (variable). We can use the **cbind()** function to join two vectors into a matrix such as **cbind(vectior1, vector2)**.

```
> x<-c(1,2,4.5,7,9)
> y<-c(0.3,0.2,0,4,5)
> z<-cbind(x,y)
> z
  x y
[1,] 1.0 0.3
[2,] 2.0 0.2
[3,] 4.5 0.0
[4,] 7.0 4.0
[5,] 9.0 5.0
```

In the above case, both x and y are vectors while z is a matrix. Sometimes we need to know how many data points (values) a vector contains (i.e., the length of a vector). In those cases, the **length()** function is used.

```
> x<-c(1:20,4:22,9)
> n<-length(x)
> n
[1] 40
```