uRProgramming

By

Zekai Otles

Created on Sun Mar 06 02:37:59 2016

Table of Contents

1. Introduction

2 . Environment Setup for R programming Developers

2.1 Environment setup

	Package	Version
1	uRProgramming	1.1.1
2	tcltk	3.2.3
3	log4r	0.2
4	odfWeave	0.8.4
5	XML	3.98-1.3
6	lattice	0.20-33
7	chron	2.3-47
8	stats	3.2.3
9	graphics	3.2.3
10	grDevices	3.2.3
11	utils	3.2.3
12	datasets	3.2.3
13	methods	3.2.3
14	base	3.2.3

```
> environment("ls")
usage of ls()
functNameusage of ls(all=TRUE)
functName
> cat("Current Working Dir \n", getwd())
Current Working Dir
C:/Users/zekai/Documents/rstudio_projects/odfWeaveTmp
```

2.2 Using Version Control System

2.3 Using IDE to develop R scripts

2.4 Writing functions

2.5 Create R package System

3. Simple Data Types and Objects

4. Control Structures

Control structures are used to change a flow or follow a logic. The control structures are easily could make any program more followable for maintenance and understanding, they could also turn the most unfollowable code. Control structures are very straight forward when used in a modular way in a less number of statements or calling other reusable functions.oo

In the following sections, the usage of *if* block will be explained with examples. The samples should help reader to solidify the understandif of te control structures.

4.1 If statements

Let's use simple vector which contains only numbers, check if numbers is equal to specific value assign iFound as True.

If (Statements) { Assignment}

```
x<-c(0,3,6,9,12,15,18,21,24)
if(x[which(x==6)]==6){iFound<-TRUE}
print(iFound)
```

In the above example, which finds the index number for the value. The usage of *which* in vector is pretty powerful as shown in the example. The *which* statement is pretty handy determining the index of vector or array. When the vector with right index with exact value found, iFound assigned to be *true*.

Logical operators in R are

Operator	Description
==	Equal to
!=	Not equal to
<	Less than
<=	Less or equal to
>	Greater than
>=	Greater or equal to
&	And
|	Or

Following code snippet is simple example of how to use logical operators. The first example is the exacty *equal* to operator. The second example demonstrates *less than or equal* to operator and assigning the value based on the result. The third example shows *greater than* comparison.The last example is combination *greater than, and, less then or equal* to.

```
Ex. 1
if(demDat[indx,1] ==1 ) {demDat[indx,1]<-'M'}
```

```
Ex.2
if(demDat[indx,2] <=25 )demDat[indx,2]<-'Y'

Ex.3
if(demDat[indx,2] > 50 ) {demDat[indx,2]<-'O'}

Ex.4
if(demDat[indx,2] > 25 & demDat[indx,2] <=50 )demDat[indx,2]<-'M'
```

4.2 If then Else statements

If then else block is used when comparison expected to result in a choice either true or false. It gives the programmer, the power to control logical flow. In R programming, if then else blocks are in the following format

If (Statements) { Assignment}

else (statements) { Assignment}

4.3 Switch statements

5. Loop Structures

The loop structures are needed to access elements of vectors, matrix, list and data frame. The access of the data is important to reach or manipulate the elements. The *for* and *while* loop are used to iterate elements.

5.1 *For loop*

The for loop is used for sequentially access the data elements. Each element of data stuctures can be reached, used or manipulated within the *for* loop. The *for* loop is in the following format.

for (values in sequence) {

    statement

}

The following code snippets how *for* loop is used. In the following example, 100 number with specified seed assign to vector. This vector has only integers for demonstration pupose.

```
set.seed(312)
#create 100 random number between 0 and 1000
xInt<-floor(runif(100,0,1000))
for(indx in 0:length(xInt) ){
if(indx %%10 ==0){print(xInt[indx])}
}
numeric(0)
[1] 450
[1] 261
[1] 405
[1] 295
[1] 763
[1] 356
[1] 833
[1] 329
[1] 419
[1] 948
```

5.2 *While loop*

The access of certain elements could be controlled by using while loop, the expression determines executing statements or not.

while(expression){

    statement

}

In the following example, the elements between two indices are assigned as either even or odd. Execution will stop when expression is false, otherwise statements within the block is executed.

```
set.seed(312)
```

```
#create 100 random number between 0 and 1000
xInt<-floor(runif(100,0,1000))
index<-80
while(index<=length(xInt)){
xInt[index]<-ifelse(as.integer(xInt[index]) %%2==0,"Even","Odd")
if(index %%4 ==0){print(xInt[index])}
index<-index+1
}
[1] "Odd"
[1] "Even"
[1] "Odd"
[1] "Odd"
[1] "Odd"
[1] "Even"
```

6. Data Structures

The main R structures are either vector (array), matrix, data frame and list. The usage and examples how to use data stuctures will help reader to access data to manipulate and display.

6.1 Vector

The vector or one dimensional array are extremely useful data structure in R. We can easily store continous and categorical data in a vector. In general, the iterator method is used to reach elements of vector to manipulate or use. The R has built in methods in *base-package* such as *c, which, sum, append, rev* and *sort* are some of the functions. Here, we will explain and the usage of some of the function with vector data structures. The first example shows how to store strings into a vector and use sort function to alphebatically store into the same vector and print out the vector. The second example is creating random number generated vector elements. Select only subset of elements (based on index number are modulus of 10) and asssign the values to new vector and find the sum of it. The loop structure is also used to demonstrate how to reach elements of the vector.

```
Ex. 1
Assign a vector with text fields
print(xText)
[1] "Elma"     "Portakal" "Armut"    "Erik"     "Seftali"  "Kaysi"
sort the vector and reassign to itself
xText<-sort(XText)
[1] "Armut"    "Elma"     "Erik"     "Kaysi"    "Portakal" "Seftali"
print(xText)

Ex. 2
set.seed(312)
#create 100 random number between 0 and 1000
xInt<-floor(runif(100,0,1000))
Find the vector elements has modulus of 10
indicies<-which(xInt%%10 ==0)
print(indicies)
[1]  9 10 15 16 22 34 56 81
The usage of for loop is shown previously.
xNewInt<-array('',dim=length(indicies))
xNewInt<-as.numeric(xNewInt)
for(indx in 1:length(indicies) ){
xNewInt[indx]<-xInt[indicies[indx]]
}
sum(xNewInt)
[1] 4310
```

6.2 Matrix

6.3 DataFrame

6.4 List

7. Simple Statistical Analysis

8. Simple Statistical Analysis

9. Simple Table Outputs

References