uRProgramming

By

Zekai Otles
Created on Sun Mar 20 23:02:06 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

Talk about version control system, why we use git system. Briefly explains the benefits of git.
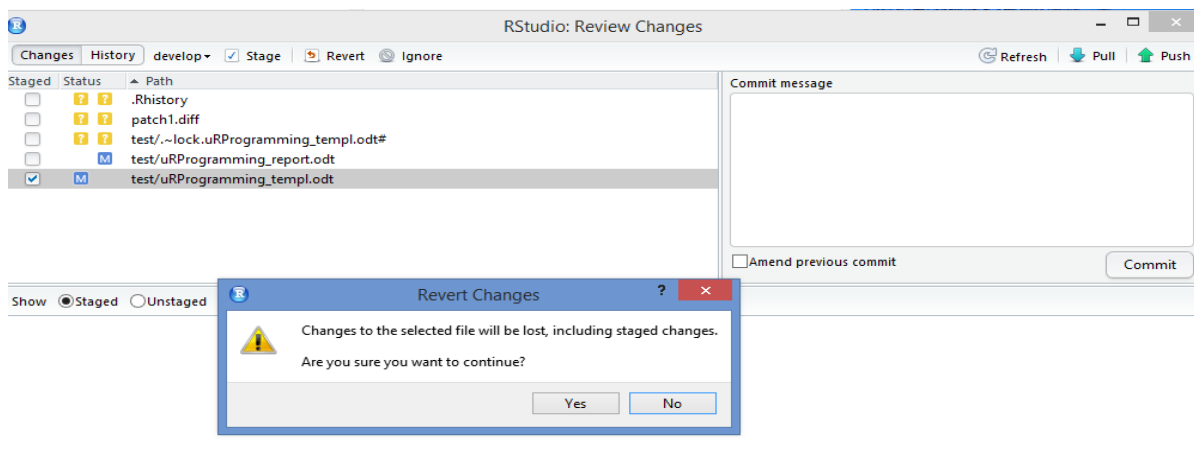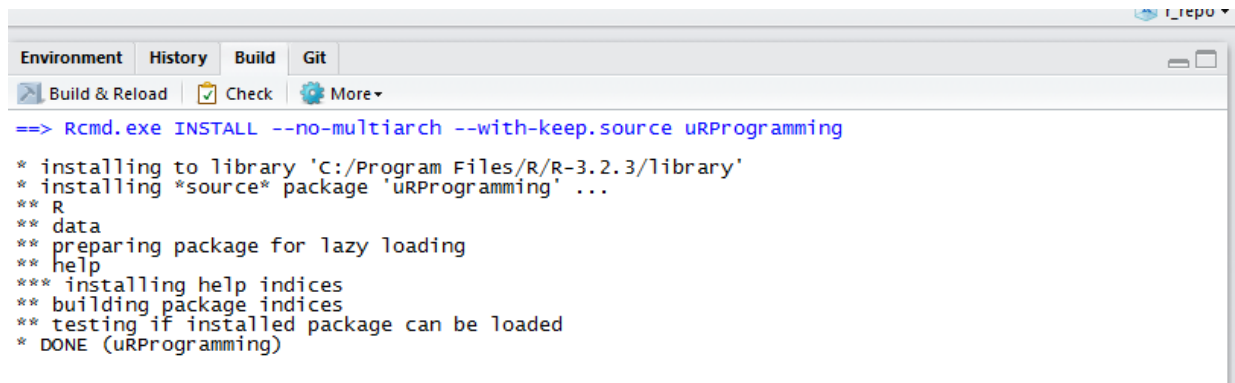
2.3 Using IDE to develop R scripts

Rstudio is our choice of integrated development environment (IDE). It allows to use modern software tools like git version control system. The Rstudio is also integrated with rapid build tools for the packages. Use figures from open office
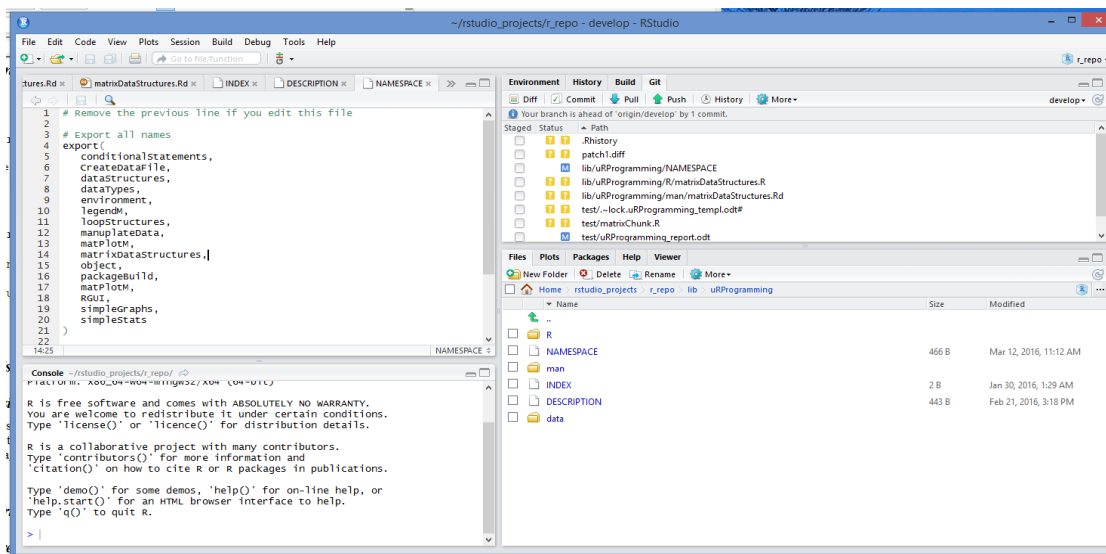
2.4 Writing Functions

2.5 Create R package System

```
r_repo ▾

Environment   History   Build   Git                                            ▬ ☐

 Build & Reload | ☑ Check | ⚙ More▾

==> Rcmd.exe INSTALL --no-multiarch --with-keep.source uRProgramming

* installing to library 'C:/Program Files/R/R-3.2.3/library'
* installing *source* package 'uRProgramming' ...
** R
** data
** preparing package for lazy loading
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded
* DONE (uRProgramming)
```

2.6   Open Office as an Integrator

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 *whil*e 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 is 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, 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 (every 10th element), 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

Matrix is two dimensional array. Matrix is used to store rectangular and table data. Any time, one has a data in a spreadsheet, table or regular form of data, the matrix data sructure should be one of the choices. In order to reach elements of matrix, the indices of matrix is used.

In following examples will show how to use matrix data structure to reach and manipulate the data. In the first example, we created simple one dimensional array and parse elements and created matrix to print in a table format. In the second example, we read a simple gridded data and reach each of the elements using loop structures for the demonstration purposes. In third same example, some of basic functions like *which*, *colSums*, and *colMeans* from base R library is used.

```
Ex. 1
set.seed(312)
#create 100 random number between 0 and 1000
xInt<-floor(runif(100,0,1000))
colAVals<-as.integer(xInt[1:10])
colBVals<-as.integer(xInt[25:35])
matr<-matrix(c(colAVals,colBVals),nrow=length(colAVals),ncol =
2,dimnames=list(NULL,c("colA","colB"))
return(matr)
```

colA	colB
791	185
661	235
934	678
451	317
819	405
669	751
228	329
426	816
460	870
450	131

```
Ex. 2


data(mtcars)
mpgVals<-array("",dim=dim(mtcars)[1]
for (rowIndx in 2:dim(mtcars)[1]){
for (colIndx in 1:dim(mtcars)[2]){
if(colIndx ==1){mpgVals[rowIndx-1]<-mtcars[rowIndx,2]}
}

print every 4th element
print(mpgVals[seq(1,length(mpgVals),4)])
[1] "6" "6" "6" "8" "4" "8" "4" "6"
matr<-as.matrix(mtcars)
```

The following *mtcars* data from R baseline will be used to demonstrate some of avaliable functions for the matrix data structures.

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.000	6.000	160.000	110.000	3.900	2.620	16.460	0.000	1.000	4.000	4.000
Mazda RX4 Wag	21.000	6.000	160.000	110.000	3.900	2.875	17.020	0.000	1.000	4.000	4.000
Datsun 710	22.800	4.000	108.000	93.000	3.850	2.320	18.610	1.000	1.000	4.000	1.000
Hornet 4 Drive	21.400	6.000	258.000	110.000	3.080	3.215	19.440	1.000	0.000	3.000	1.000
Hornet Sportabout	18.700	8.000	360.000	175.000	3.150	3.440	17.020	0.000	0.000	3.000	2.000
Valiant	18.100	6.000	225.000	105.000	2.760	3.460	20.220	1.000	0.000	3.000	1.000
Duster 360	14.300	8.000	360.000	245.000	3.210	3.570	15.840	0.000	0.000	3.000	4.000
Merc 240D	24.400	4.000	146.700	62.000	3.690	3.190	20.000	1.000	0.000	4.000	2.000
Merc 230	22.800	4.000	140.800	95.000	3.920	3.150	22.900	1.000	0.000	4.000	2.000
Merc 280	19.200	6.000	167.600	123.000	3.920	3.440	18.300	1.000	0.000	4.000	4.000
Merc 280C	17.800	6.000	167.600	123.000	3.920	3.440	18.900	1.000	0.000	4.000	4.000
Merc 450SE	16.400	8.000	275.800	180.000	3.070	4.070	17.400	0.000	0.000	3.000	3.000
Merc 450SL	17.300	8.000	275.800	180.000	3.070	3.730	17.600	0.000	0.000	3.000	3.000
Merc 450SLC	15.200	8.000	275.800	180.000	3.070	3.780	18.000	0.000	0.000	3.000	3.000

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Cadillac Fleetwood	10.400	8.000	472.000	205.000	2.930	5.250	17.980	0.000	0.000	3.000	4.000
Lincoln Continental	10.400	8.000	460.000	215.000	3.000	5.424	17.820	0.000	0.000	3.000	4.000
Chrysler Imperial	14.700	8.000	440.000	230.000	3.230	5.345	17.420	0.000	0.000	3.000	4.000
Fiat 128	32.400	4.000	78.700	66.000	4.080	2.200	19.470	1.000	1.000	4.000	1.000
Honda Civic	30.400	4.000	75.700	52.000	4.930	1.615	18.520	1.000	1.000	4.000	2.000
Toyota Corolla	33.900	4.000	71.100	65.000	4.220	1.835	19.900	1.000	1.000	4.000	1.000
Toyota Corona	21.500	4.000	120.100	97.000	3.700	2.465	20.010	1.000	0.000	3.000	1.000
Dodge Challenger	15.500	8.000	318.000	150.000	2.760	3.520	16.870	0.000	0.000	3.000	2.000
AMC Javelin	15.200	8.000	304.000	150.000	3.150	3.435	17.300	0.000	0.000	3.000	2.000
Camaro Z28	13.300	8.000	350.000	245.000	3.730	3.840	15.410	0.000	0.000	3.000	4.000
Pontiac Firebird	19.200	8.000	400.000	175.000	3.080	3.845	17.050	0.000	0.000	3.000	2.000
Fiat X1-9	27.300	4.000	79.000	66.000	4.080	1.935	18.900	1.000	1.000	4.000	1.000

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Porsche 914-2	26.000	4.000	120.300	91.000	4.430	2.140	16.700	0.000	1.000	5.000	2.000
Lotus Europa	30.400	4.000	95.100	113.000	3.770	1.513	16.900	1.000	1.000	5.000	2.000
Ford Pantera L	15.800	8.000	351.000	264.000	4.220	3.170	14.500	0.000	1.000	5.000	4.000
Ferrari Dino	19.700	6.000	145.000	175.000	3.620	2.770	15.500	0.000	1.000	5.000	6.000
Maserati Bora	15.000	8.000	301.000	335.000	3.540	3.570	14.600	0.000	1.000	5.000	8.000
Volvo 142E	21.400	4.000	121.000	109.000	4.110	2.780	18.600	1.000	1.000	4.000	2.000

In the Ex. 3, *which* is allowing us to assign the row to the vector when the rowname of matrix is "Merc 450SE"; see the code snippet with "*matrx[which(rownames(matrx)=="Merc 450SE"),]*". The *colSums* finds the sum of each colums, there is also *rowSums* which finds the sum of each row. The *colMeans* is average values for each column, the *rowMeans* yields the average values for each row.

Ex. 3

```
data(mtcars)
matrx<-as.matrix(mtcars)
merc<-matrx[ which(rownames(matrx)=="Merc 450SE"),]
print(merc)
colSums(matr)
colMeans(matr)
matr<-rbind(colSums(matrx),colMeans(matrx))
```

6.3 DataFrame

The dataframe is used frequently in R. The different type of data can be stored and retrieved. In simple example, 4 vector (ad, soyad, okulNo, ve kayitTarihi) will be stored in dataframe (df).

```
ad<-c('Ali','Ahmet','Fatma','Ayse')
soyad<-c('Kasap','Berber','Kaya','Gul')
okulNo<-as.integer(c(11,23,5,100))
kayitTarihi<-as.Date(c('1979-11-1','1980-5-4','1979-5-25','1981-10-10'))
df<-data.frame(ad,soyad,okulNo,kayitTarihi)
structure(df)
      ad  soyad okulNo kayitTarihi
1   Ali  Kasap     11  1979-11-01
2 Ahmet Berber     23  1980-05-04
3 Fatma   Kaya      5  1979-05-25
4  Ayse    Gul    100  1981-10-10
```

6.4 List

7. Simple Statistical Analysis

8. Simple Statistical Analysis

9. Simple Table Outputs

References