R Tutorial Sample Code

Yuhui Zhang

March 25, 2015

Contents

Introduce R

R is vectorized:

```
A = c(1, 2, 3)
B = c(11, 22, 33)
f = A * B
f
```

```
## [1] 11 44 99
```

R Data Types

Hello world

Say hello to R:

```
c <- "Hello World" # type Enter
print(c) # print some text, anything after a hash (#) is a comment
```

```
## [1] "Hello World"
```

Basic data types (dates and times)

Assignment operator

Assignment operator:

```
n <- 10   # assign value 10 to variable 'n'
0.3 -> s  # the arrow can go both ways
m = TRUE   # can also use equal (=) operator for assignment
m
```

```
## [1] TRUE
```

Dates and times

Simple example of date:

```
x <- as.Date("2015-03-26")
x
```

```
## [1] "2015-03-26"
```

A little complicated example of time (POSIXlt):

```r
x <- Sys.time()
x
```

```
## [1] "2015-03-29 18:05:21 PDT"
```

```r
p <- as.POSIXlt(x) # unclass(p) is a list object
names(unclass(p))
```

```
##  [1] "sec"    "min"    "hour"   "mday"   "mon"    "year"   "wday"
##  [8] "yday"   "isdst"  "zone"   "gmtoff"
```

```r
p$sec
```

```
## [1] 21.10242
```

strptime function (from characters to POSIXlt):

```r
timeString <- "March 26, 2015 12:30"
x <- strptime(timeString, "%B %d, %Y %H:%M")
class(x)
```

```
## [1] "POSIXlt" "POSIXt"
```

```r
x
```

```
## [1] "2015-03-26 12:30:00 PDT"
```

Compound objects

Vector

Define an empty vector:

```r
x  <-  vector("numeric",  length  =  10)
```

Define vector using c():

```r
x  <-  c(0.5,  0.6) #number
x  <-  c(TRUE,  FALSE) # logical
x  <-  c(T,  F) #logical
x  <-  c("a",  "b",  "c") #character
x  <-  c(1+0i,  2+4i) #complex
```

Colon operator (create a sequence of numbers):

```r
x = 1:20
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

List

First example of list:

```r
x  <-  list(1,  "a",  TRUE,  1  +  4i)
x
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 1+4i
```

Second example of list:

```r
x <- list(a=c(T,T,F,F), b=2)
x
```

```
## $a
## [1]  TRUE  TRUE FALSE FALSE
##
## $b
## [1] 2
```

Factor

Define a factor:

```r
x  <-  factor(c("yes",  "yes",  "no",  "yes",  "no"))
x
```

```
## [1] yes yes no  yes no
## Levels: no yes
```

```r
table(x)
```

```
## x
##  no yes
##   2   3
```

Change the sequence of factor's levels:

```
x  <-  factor(c("yes",  "yes",  "no",  "yes",  "no"))
x
```

```
## [1] yes yes no  yes no
## Levels: no yes
```

```
x  <-  factor(x,levels=c("no","yes"))
x
```

```
## [1] yes yes no  yes no
## Levels: no yes
```

Matrix

Define an empty matrix:

```
m  <-  matrix(nrow  =  2,  ncol  =  3)
m
```

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

```
dim(m)
```

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

```
attributes(m)
```

```
## $dim
## [1] 2 3
```

Matrix is column-wise constructed:

```
m  <-  matrix(1:6,  nrow  =  2,  ncol  =  3)
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

cbind() and rbind() for matrix:

```
x  <-  1:3
y  <-  10:12
cbind(x,y)
```

```
##      x  y
## [1,] 1 10
## [2,] 2 11
## [3,] 3 12
```

```
rbind(x,y)
```

```
##   [,1] [,2] [,3]
## x    1    2    3
## y   10   11   12
```

Naming the matrix:

```
m <- matrix(1:4, nrow = 2, ncol = 2)
dimnames(m)
```

```
## NULL
```

```
dimnames(m) <- list(c("a", "b"), c("c", "d"))
m
```

```
##   c d
## a 1 3
## b 2 4
```

Data Frame

Define a simple data frame:

```
x <- data.frame(foo = 1:4, bar = c(T, T, F, F))
x
```

```
##   foo   bar
## 1   1  TRUE
## 2   2  TRUE
## 3   3 FALSE
## 4   4 FALSE
```

```
nrow(x)
```

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

```
ncol(x)
```

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

Coercion

Coercion as vector:

```r
y <- c(1.7, "a") #character
y
```

```
## [1] "1.7" "a"
```

```r
y <- c(TRUE, 2) #numeric
y
```

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

Explicit coercion

```r
x <- 0:6
class(x)
```

```
## [1] "integer"
```

```r
as.numeric(x)
```

```
## [1] 0 1 2 3 4 5 6
```

```r
as.logical(x)
```

```
## [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```r
as.character(x)
```

```
## [1] "0" "1" "2" "3" "4" "5" "6"
```

```r
as.complex(x)
```

```
## [1] 0+0i 1+0i 2+0i 3+0i 4+0i 5+0i 6+0i
```

Missing values

NA and NaN

```r
x <- c(1, 2, NA, 10, 3)
is.na(x)
```

```
## [1] FALSE FALSE  TRUE FALSE FALSE
```

```r
is.nan(x)
```

```
## [1] FALSE FALSE FALSE FALSE FALSE
```

```r
x <- c(1, 2, NaN, NA, 4)
is.na(x)
```

```
## [1] FALSE FALSE  TRUE  TRUE FALSE
```

```r
is.nan(x)
```

```
## [1] FALSE FALSE  TRUE FALSE FALSE
```

Basic Operations

Subsetting

In vector and matrix

```r
x <- c("a", "b", "c", "c", "d", "a")
x[2]
```

```
## [1] "b"
```

```r
x[1:4]
```

```
## [1] "a" "b" "c" "c"
```

```r
x > "a"
```

```
## [1] FALSE  TRUE  TRUE  TRUE  TRUE FALSE
```

```r
x[x>"a"]
```

```
## [1] "b" "c" "c" "d"
```

```r
x <- matrix(1:6, 2, 3)
x[1,2]
```

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

```r
x[1,] # Entire first row.
```

```
## [1] 1 3 5
```

```r
x[,2] # Entire second column.
```

```
## [1] 3 4
```

In matrix (the exception of []):

```r
x <- matrix(1:6, 2, 3)
x[1,2]
```

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

```r
x[1,2,drop=FALSE]
```

```
##      [,1]
## [1,]    3
```

```r
x[1,]
```

```
## [1] 1 3 5
```

```r
x[1,,drop=FALSE]
```

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

In list:

```r
x <- list(foo = 1:4, bar = 0.6)
x[1]
```

```
## $foo
## [1] 1 2 3 4
```

```r
x$foo
```

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

```r
x$bar
```

```
## [1] 0.6
```

```r
x["bar"]
```

```
## $bar
## [1] 0.6
```

```r
x[["bar"]]
```

```
## [1] 0.6
```

Extracting multiple elements of a list:

```r
x <- list(foo = 1:4, bar = 0.6, baz = "hello")
x[c(1, 3)]
```

```
## $foo
## [1] 1 2 3 4
##
## $baz
## [1] "hello"
```

Partial matching

```r
x <- list(addedName = 1:5)
x$a
```

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

```r
x[["a"]]
```

```
## NULL
```

```r
x[["a",exact=FALSE]]
```

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

Control structures

For-loop:

```r
for(i in 1:10) {
   print(i*i)
}
```

```
## [1] 1
## [1] 4
## [1] 9
## [1] 16
## [1] 25
## [1] 36
## [1] 49
## [1] 64
## [1] 81
## [1] 100
```

While-loop:

```r
i=1
while(i<=10) {
   print(i*i)
   i=i+sqrt(i)
}
```

```
## [1] 1
## [1] 4
## [1] 11.65685
## [1] 27.68836
## [1] 57.0912
```

Repeat-loop:

```
i=1
repeat {
 i= i+1
 if (i>=10) break # the only way to get out of the loop
}
```

Build-in functions

User written function

User-written function:

```
foo <- function(x,y=1,...) {
  cat("extra args:",...,"\n")
  sqrt(x)+sin(y)
}
foo(1,2,3,"bar")
```

```
## extra args: 3 bar

## [1] 1.909297
```

```
foo(1)
```

```
## extra args:

## [1] 1.841471
```

```
foo(y=3,x=7)
```

```
## extra args:

## [1] 2.786871
```

```
foo
```

```
## function(x,y=1,...) {
##   cat("extra args:",...,"\n")
##   sqrt(x)+sin(y)
## }
```

str() function:

```
str(lm)
```

```
## function (formula, data, subset, weights, na.action, method = "qr",
##     model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
##     contrasts = NULL, offset, ...)
```

"*apply"

lapply and sap ply

lapply:

```
lapply(c( -1, -5), abs)
```

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

sapply:

```
sapply(c( -1, -5), abs)
```

```
## [1] 1 5
```

apply

apply:

```
x <- matrix(c(1, 2, 3, 4), 2, 2)
x
```

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

```
apply(x, 1, sum)
```

```
## [1] 4 6
```

```
apply(x, 2, sum)
```

```
## [1] 3 7
```

tapply

tapply

```r
score <- c(90, 79, 94, 85)
gender <- factor(c("Male", "Female", "Female", "Male"))
tapply(score, gender, mean)
```

```
## Female   Male
##   86.5   87.5
```

mapply

mapply:

```r
mapply(rep, c(0,2), c(3,5)) # input certain combinations
```

```
## [[1]]
## [1] 0 0 0
##
## [[2]]
## [1] 2 2 2 2 2
```

Packages

working directory and R script

Reading and Writing Data

R/W local flat files

Read local flat file (read.csv):

```r
titanic_data <- read.csv("Titanic.csv") # put Titanic.csv in your local directory
head(titanic_data, 3)
```

```
##   X Class  Sex   Age Survived Freq
## 1 1   1st Male Child       No    0
## 2 2   2nd Male Child       No    0
## 3 3   3rd Male Child       No   35
```

Read local flat file (readLines):

```r
line1 <- readLines("Titanic.csv", 1)
line1
```

```
## [1] "\"\",\"Class\",\"Sex\",\"Age\",\"Survived\",\"Freq\""
```

R/W local Excel files

Read local Excel file:

```r
# Install the xlsx library
# install.packages('xlsx')
# Load the library
library(xlsx)
# Now you can Read the Excel file
titanic_data <- read.xlsx("titanic3.xls", sheetIndex=1)
head(titanic_data, 3)
```

```
##   pclass survived                          name    sex     age sibsp
## 1      1        1  Allen, Miss. Elisabeth Walton female 29.0000     0
## 2      1        1 Allison, Master. Hudson Trevor   male  0.9167     1
## 3      1        0   Allison, Miss. Helen Loraine female  2.0000     1
##   parch ticket     fare   cabin embarked boat body
## 1     0  24160 211.3375      B5        S    2 <NA>
## 2     2 113781 151.5500 C22 C26        S   11 <NA>
## 3     2 113781 151.5500 C22 C26        S <NA> <NA>
##                         home.dest
## 1                     St Louis, MO
## 2 Montreal, PQ / Chesterville, ON
## 3 Montreal, PQ / Chesterville, ON
```

Connection interfaces

Simple example 1:

```r
con <- file("Titanic.csv", "r")
titanic_data <- read.csv(con)
close(con)
```

Simple example 2:

```r
con <- url("http://vincentarelbundock.github.io/Rdatasets/csv/datasets/Titanic.csv")
another_data <- read.csv(con)
```

Reading XML/HTML files

Example:

```r
library(XML) # you need to install this library
url <- "http://www.w3schools.com/xml/simple.xml"
doc <- xmlTreeParse(url, useInternal=TRUE)# also works for html
rootNode <- xmlRoot(doc)
rootNode[[1]]
```

```
## <food>
##   <name>Belgian Waffles</name>
##   <price>$5.95</price>
##   <description>Two of our famous Belgian Waffles with plenty of real maple syrup</description>
##   <calories>650</calories>
## </food>
```

```
rootNode[[1]][[1]]
```

```
## <name>Belgian Waffles</name>
```

```
xpathSApply(rootNode, "//name", xmlValue)
```

```
## [1] "Belgian Waffles"             "Strawberry Belgian Waffles"
## [3] "Berry-Berry Belgian Waffles" "French Toast"
## [5] "Homestyle Breakfast"
```

R/W to JSON

Example:

```
library(jsonlite)
jsonData <- fromJSON("http://citibikenyc.com/stations/json")
names(jsonData)
```

```
## [1] "executionTime"   "stationBeanList"
```

```
jsonData$stationBeanList[1,1:3]
```

```
##   id      stationName availableDocks
## 1 72 W 52 St & 11 Ave             36
```

```
head(iris, 3)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
```

```
iris2 <- toJSON(iris, pretty=TRUE)
```

Connect to a database

Example:

```
library(RMySQL)# load the library
ucscDb <- dbConnect(MySQL(), user="genome", host="genome-mysql.cse.ucsc.edu")
data <- dbGetQuery(ucscDb, "show databases;") # get the output of SQL query as data frame in R
head(data)
```

```
##             Database
## 1 information_schema
## 2            ailMel1
## 3            allMis1
## 4            anoCar1
## 5            anoCar2
## 6            anoGam1
```

```
dbDisconnect(ucscDb)# don't forget to close the connection
```

```
## [1] TRUE
```

Textual format

Statistical Simulation

Example of normal distribution:

```
dnorm(0, mean = 0, sd = 1)
```

```
## [1] 0.3989423
```

```
dnorm(10, mean = 0, sd = 1)
```

```
## [1] 7.694599e-23
```

```
pnorm(0, mean = 0, sd = 1)
```

```
## [1] 0.5
```

```
pnorm(1, mean = 0, sd = 1)
```

```
## [1] 0.8413447
```

```
pnorm(100, mean = 0, sd = 1)
```

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

```
qnorm(0.5, mean = 0, sd = 1)
```

```
## [1] 0
```

```
qnorm(0, mean = 0, sd = 1)
```

```
## [1] -Inf
```

```
qnorm(0.2, mean = 0, sd = 1)
```

```
## [1] -0.8416212
```

```r
rnorm(4, mean = 0, sd = 1)
```

```
## [1]  1.5952808  0.3295078 -0.8204684  0.4874291
```

Reproducible random numbers

```r
rnorm(3, mean = 0, sd = 1)
```

```
## [1]  0.7383247  0.5757814 -0.3053884
```

```r
set.seed(1)
rnorm(3, mean = 0, sd = 1)
```

```
## [1] -0.6264538  0.1836433 -0.8356286
```

```r
set.seed(1)
rnorm(3, mean = 0, sd = 1)
```

```
## [1] -0.6264538  0.1836433 -0.8356286
```