R Programmming 101 (Beginner Tutorial)

MFSA / Stats Club

Last Updated: June 24, 2015

R for Statisitical Computing

Download R from `http://cran.r-project.org/mirrors.html`

Recommended supplement but not necessary: RStudio from `http://www.rstudio.com/products/rstudio/download/`

- R Studio: A (very matlab like) IDE for R

Comments

Comments are typed with hashtags '#'

```
# This is a comment
cat("This is not a comment")
```

```
## This is not a comment
```

No block comments. So sad. =(

Data Types

Integers & Numerics

Examples: 1,2.0,1.1,pi

```
c(1,2.0,1.1,pi)
```

```
## [1] 1.000000 2.000000 1.100000 3.141593
```

- `Inf` can also be used in calculations

```
1/Inf
```

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

Complex Numbers

We can even use complex numbers.

```
complex(real = 1, imaginary = 2)
```

```
## [1] 1+2i
```

```
8+6i
```

```
## [1] 8+6i
```

Characters

Example: 'One', '1', 'pi'

```
c('One', '1', 'pi')
```

Boolian (Logical) Values

Boolian values can take only two values: `TRUE` (`T`) or `FALSE` (`F`).

```
c(TRUE, FALSE, TRUE)
```

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

Factors

A factor is a categorical variable that can take on only a finite set of values. i.e. Sex, Faculty in University

```
factor(c('Male','Female','Male','Male'))
```

```
## [1] Male   Female Male   Male
## Levels: Female Male
```

Everything is an object, including functions!

Vectors

Most common objects are called vectors.
Examples: vector of numbers

```
a1 <- c(1,2,3)
a1
```

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

```
a2 <- c('one','two','three')
a2
```

```
## [1] "one"   "two"   "three"
```

```r
a3 <- c('1','2','3')
a3
```

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

You can also create a range of values using `start:end`

```r
4:10
```

```
## [1]  4  5  6  7  8  9 10
```

```r
4:-3
```

```
## [1]  4  3  2  1  0 -1 -2 -3
```

```r
0.1:4
```

```
## [1] 0.1 1.1 2.1 3.1
```

Basic Numerical Operations: +, -, *, /, ^

Numerical operations are: +, -, *, /, ^

- These operate elementwise between vectors.

Operator	Description
+	Addition
-	Subtraction
	Multiplication
/	Division
^	Power

```r
c(1,2,3) * c(4,5,6)
```

```
## [1]  4 10 18
```

Note: They don't have to have the same length. If they don't then the vector will recycle though the shorter vector. The longer has to be a multiple of the shorter vector.

```r
c(1,2,3) ^ c(1,2,3,4,5,6)
```

```
## [1]   1   4  27   1  32 729
```

Logical Operators

- Return a boolan value of `TRUE` or `FALSE`

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

Operator	Description
&	Elementwise And
||	Or
&&	And

```
c(TRUE, FALSE) | c(FALSE, FALSE)
```

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

```
c(1,2,3) < c(2,1,4)
```

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

Pro Tip: When interacting with number, boolians are converted to an integer: 0, or 1.

Type check

`is.(typename)`

Example: `is.vector`, `is.integer`, `is.numeric`, `is.data.frame`, `is.matrix`

- Not sure which type? Use `typeof()` !

```
is.numeric(a1)
```

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

```
is.vector(a1)
```

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

```
is.data.frame(a1)
```

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

Assignment Operator

Assignment can come in 3 forms:

```
var_name <- evaluation
var_name = evaluation
evaluation -> var_name
```

```
x <- 1
x
```

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

Be careful: <- is not the same as < -

```
x < -1
```

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

```
y = "string"
y
```

```
## [1] "string"
```

```
"This isn't used much" -> z
z
```

```
## [1] "This isn't used much"
```

Concatenating Vectors

They are different vectors! To concatenate two vectors, use `c(vector.1, vector.2)`

```
b12 <- c(a1,a2)
b12
```

```
## [1] "1"     "2"     "3"     "one"   "two"   "three"
```

```
b23 <- c(a2,a3)
b23
```

```
## [1] "one"   "two"   "three" "1"     "2"     "3"
```

```
b13 <- c(a1,a3)
b13
```

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

```
b21 <- c(a2,a1)
b21
```

```
## [1] "one"   "two"   "three" "1"     "2"     "3"
```

Notice that when combined with characters, numerics are changed into characters automatically. So `b23 == b21`.

```
b23 == b21
```

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

```
b123 <- c(a1,a2,a3)
```

Dot Product

To use dot product of two vectors (instead of elementwise) use %*%

```
a1 %*% a1
```

```
##      [,1]
## [1,]   14
```

```
c(1,4,5) %*% c(6,7,2)
```

```
##      [,1]
## [1,]   44
```

Exercise

1. What are the datatypes available in R?
2. What datatype would the vector `c(1,2,"three")` be?
3. What is the vector `c(3,4,5,6)` to the power of 4?
4. What elements of `c(3,4,5,6)` greater than 4?

Answer

1. What are the datatypes available in R?

- Numeric
- Integer
- Complex
- Character
- Boolian
- Factor

2. What datatype would the vector `c(1,2,"three")` be?

- Character

```
class(c(1,2,"three"))
```

```
## [1] "character"
```

3. What is the vector `c(3,4,5,6)` to the power of 4?

```
c(3,4,5,6) ^ 4
```

```
## [1]   81  256  625 1296
```

4. What elements of `c(3,4,5,6)` greater than 4?

```
c(3,4,5,6) > 4
```

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

Lists

Different from vectors, they allow us to put multiple structures in a list.

- Useful when we need to store a list of objects with different datatypes

```
l12 <- list(a1,a2)
l12
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "one"   "two"   "three"
```

```
l23 <- list(a2,a3)
l23
```

```
## [[1]]
## [1] "one"   "two"   "three"
##
## [[2]]
## [1] "1" "2" "3"
```

```
l13 <- list(a1,a3)
l13
```

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

Notice they are stored in two 'different arrays'

as.vector, as.list can interchange list to vectors and vectors to list via as.vector and as.list

```
as.vector(123)
```

```
## [[1]]
## [1] "one"   "two"   "three"
##
## [[2]]
## [1] "1" "2" "3"
```

```
as.list(a1)
```

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

Exercise

1. Generate a vector of 1 to 10, a list of characters 2.1 to 2.5 separated by 0.1
2. Add the list to the vector and return a vector
3. Define 2 vectors, x1, x2, by using rnorm(7,mean $=$ 13, sd $=$ 5)
4. Do the inner product of x1,x2

Answer

```
q1 = 1:10 #Question 1
q1c = as.list(as.character(seq(2.1,2.5,0.1)))

q1 + as.numeric(q1c) # Question 2
```

```
##  [1]  3.1  4.2  5.3  6.4  7.5  8.1  9.2 10.3 11.4 12.5
```

```
x1 = rnorm(7,mean = 13, sd = 5) #Question 3
x2 = rnorm(7,mean = 13, sd = 5)

x1 %*% x2 #Question 4
```

```
##           [,1]
## [1,] 1157.286
```

Matrix

- Each column needs to contain same type.
- Like a high level vector.

```
M1 <- matrix(c(1,2,3,4,5,6,7,8,9),nrow=3,ncol=3)
M1
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
M2 <- matrix(9:1 ,3 ,3)
M2
```

```
##      [,1] [,2] [,3]
## [1,]    9    6    3
## [2,]    8    5    2
## [3,]    7    4    1
```

```
M3 <- matrix(c(a1,a2),2,3)
M3
```

```
##      [,1] [,2]  [,3]
## [1,] "1"  "3"   "two"
## [2,] "2"  "one" "three"
```

Data Frames

- Generalised matrix. Now we can store different data types in different columns! $=)$
- Like high level list

```
df1 <- data.frame(a1,a2,a3)
df1
```

```
##   a1    a2 a3
## 1  1   one  1
## 2  2   two  2
## 3  3 three  3
```

Attributes

Attribute	Description
`names`	Names of an object
`dimnames`	Names of the dimensions of an object
`dim`	Dimension of an object
`class`	Class of an object
`length`	Length of an object

```
length(a1)
```

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

```r
names(a1) = c("a","b","c")
a1
```

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

```r
names(df1) = c("var_1","var_2","var_3")
df1
```

```
##   var_1 var_2 var_3
## 1     1   one     1
## 2     2   two     2
## 3     3 three     3
```

```r
dim(M1)
```

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

Data Manipulation

Indices, just like linear algebra, for vectors, specify thy entry, and matrix row first then column.

```
a1[2] # Second entry
```

```
## b
## 2
```

```
M1[1,2] #First row second column
```

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

```
df1[2,3] # Second row third column
```

```
## [1] 2
## Levels: 1 2 3
```

```r
M1[1,] # First row
```

```
## [1] 1 4 7
```

```r
M1[,3] # Third column
```

```
## [1] 7 8 9
```

You can also Boolian values to get a subset of values:

```r
a1[a1 <= 2]
```

```
## a b
## 1 2
```

Accessing the elements of a list is slightly different. Use double `[[]]` notation:

```
l13[[1]]
```

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

Assigning names to data.frame and matrices

```
rownames(M1) <- c('Ein','Zwei','Drei')
colnames(M1) <- c('Un','Deux','Trois')
M1
```

```
##      Un Deux Trois
## Ein   1    4     7
## Zwei  2    5     8
## Drei  3    6     9
```

```r
rownames(df1) <- c('Uno','Dos','Tres')
colnames(df1) <- c('yi','er','san')
df1
```

```
##       yi    er san
## Uno    1   one   1
## Dos    2   two   2
## Tres   3 three   3
```

Adding new rows or columns into matrix or data.frame

`rbind()`: Add new row to rbind, cbind

```
M1.rbind <- rbind(M1,M1)
M1.rbind
```

```
##       Un Deux Trois
## Ein    1    4     7
## Zwei   2    5     8
## Drei   3    6     9
## Ein    1    4     7
## Zwei   2    5     8
## Drei   3    6     9
```

```r
M2.rbind <- rbind(M1,M2) # Notice the names of columns and
M2.rbind
```

```
##       Un Deux Trois
## Ein    1    4     7
## Zwei   2    5     8
## Drei   3    6     9
##        9    6     3
##        8    5     2
##        7    4     1
```

```r
M1.cbind <- cbind(M1,M1)
M1.cbind
```

```
##       Un Deux Trois Un Deux Trois
## Ein    1    4     7  1    4     7
## Zwei   2    5     8  2    5     8
## Drei   3    6     9  3    6     9
```

Calling by Column Names

```
df1$yi
```

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

```
df1$er
```

```
## [1] one   two   three
## Levels: one three two
```

```
df1$san
```

```
## [1] 1 2 3
## Levels: 1 2 3
```

Reading csv/delim files

```
read.file_type(file = "Name.file_type", header =
TRUE, sep = "")
```

Useful functions

```
attach(iris)
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Spec
## 1          5.1         3.5          1.4         0.2  set
## 2          4.9         3.0          1.4         0.2  set
## 3          4.7         3.2          1.3         0.2  set
## 4          4.6         3.1          1.5         0.2  set
## 5          5.0         3.6          1.4         0.2  set
## 6          5.4         3.9          1.7         0.4  set
```

```
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.
##  Median :5.800   Median :3.000   Median :4.350   Median
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.
##        Species
##  setosa     :50
##  versicolor:50
##  virginica :50
##
##
##
```

```
print(iris)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1            5.1         3.5          1.4         0.2
## 2            4.9         3.0          1.4         0.2
## 3            4.7         3.2          1.3         0.2
## 4            4.6         3.1          1.5         0.2
## 5            5.0         3.6          1.4         0.2
## 6            5.4         3.9          1.7         0.4
## 7            4.6         3.4          1.4         0.3
## 8            5.0         3.4          1.5         0.2
## 9            4.4         2.9          1.4         0.2
## 10           4.9         3.1          1.5         0.1
## 11           5.4         3.7          1.5         0.2
## 12           4.8         3.4          1.6         0.2
## 13           4.8         3.0          1.4         0.1
## 14           4.3         3.0          1.1         0.1
## 15           5.8         4.0          1.2         0.2
## 16           5.7         4.4          1.5         0.4
```

apply, sapply, lapply

- sapply, lapply takes in vectors/list
- sapply(lapply) returns a vector(list) of same length

WARNING: DO NOT USE ANY OF lapply OR sapply under normal circumstances

```
sapply(iris$Sepal.Width,floor)
```

```
##   [1] 3 3 3 3 3 3 3 3 2 3 3 3 3 3 4 4 3 3 3 3 3 3 3 3 3
##  [36] 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 2 2 2 3 2 2 2
##  [71] 3 2 2 2 2 3 2 3 2 2 2 2 2 2 3 3 3 2 3 2 2 3 2 2 2
## [106] 3 2 2 2 3 3 2 3 2 2 3 3 3 2 2 3 2 2 2 3 3 2 3 2 3
## [141] 3 3 2 3 3 3 2 3 3 3
```

```
lapply(iris$Sepal.width,floor)
```

```
## list()
```

```
floor(iris$Sepal.Width)
```

```
##   [1] 3 3 3 3 3 3 3 3 2 3 3 3 3 3 4 4 3 3 3 3 3 3 3 3 3
##  [36] 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 2 2 2 3 2 2 2
##  [71] 3 2 2 2 2 3 2 3 2 2 2 2 2 2 3 3 3 2 3 2 2 3 2 2 2
## [106] 3 2 2 2 3 3 2 3 2 2 3 3 3 2 2 3 2 2 2 3 3 2 3 2 3
## [141] 3 3 2 3 3 3 2 3 3 3
```

Notice that this returns same thing as sapply, so there is no reason to use sapply under most of the cases.

- `apply` is a function to apply a function to a matrix by row or column or both

```
apply(M2,1,min)# Minimum for each row
```

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

```r
apply(M2,2,min)# Minimum for each column
```

```
## [1] 7 4 1
```

```r
apply(M2,c(1,2),min)# Minimum of all entries, same as min(M
```

```
##      [,1] [,2] [,3]
## [1,]    9    6    3
## [2,]    8    5    2
## [3,]    7    4    1
```

User define functions

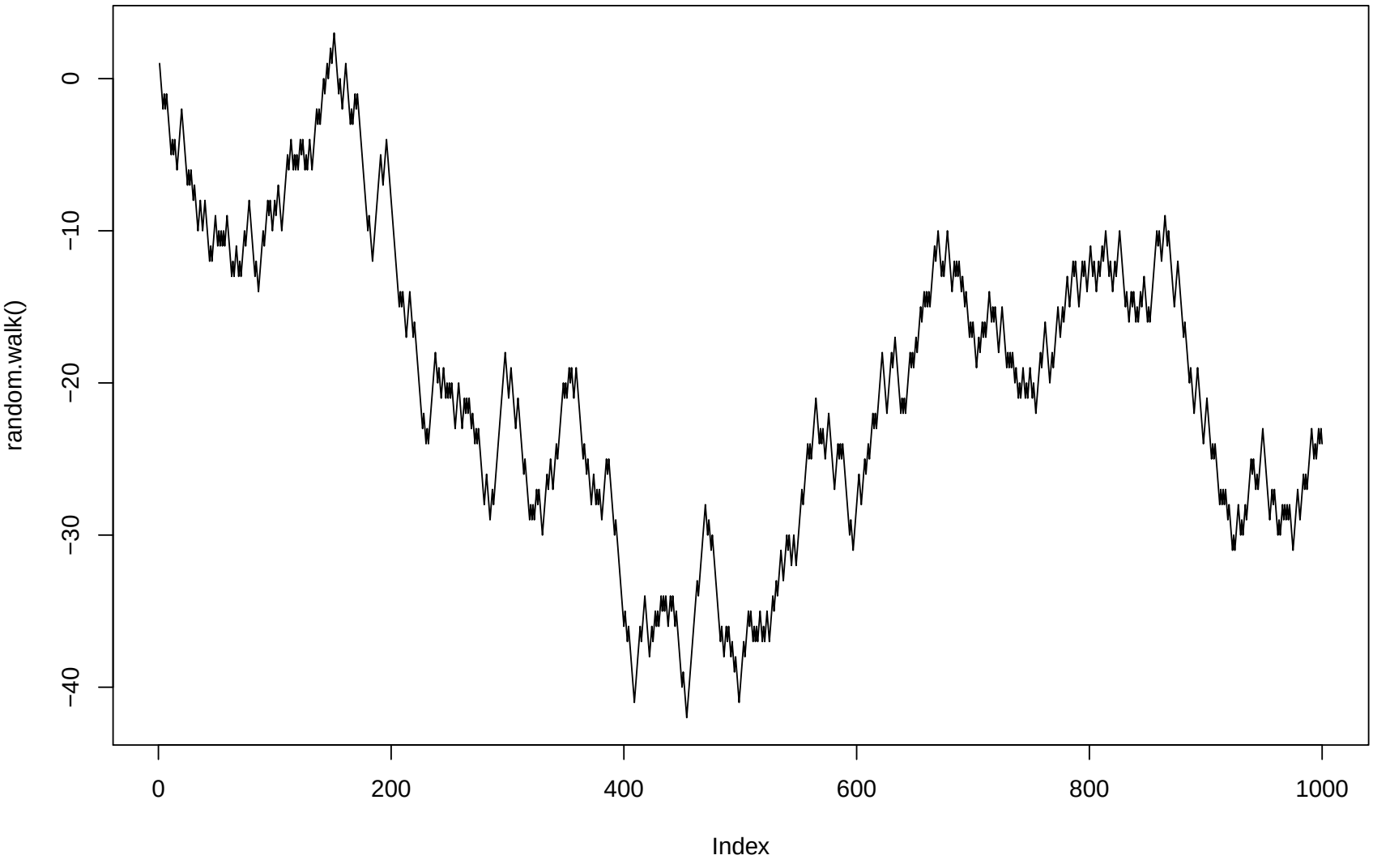
Can predefine default value for argument(s) - Can take in vectors instead of scalars

```
random.walk <- function(n=1000,p=0.5, start = 0,min = 0, ma
  rand <- runif(n = n,  min = min, max = max)/(max - min);

  steps <- sign(2*(rand - p));
  out <- start + cumsum(steps);

  return(out)
}
```

```
plot(random.walk(),type = "l")
```



Exercise

Use the iris dataset in R and build a matrix call iris.matrix with the followings:

1. Columns and rows of iris corresponds to columns and rows of iris.matrix
2. Change the Species column of iris.matrix into the following indicator variables

- 1 - setosa, 2 - versicolor, 3 - virginica

3. Get the mean of every column except for Species column
4. Take away respective mean from each column except for the Species column
5. Produce the summary of the new matrix

Futher Notes

- `browser()` is useful to help debugging, talk about this later
- `?function_name` is useful to look up what a function does

Interesting reads:

`ggplot2`: Plot package based around "the grammer of graphics"
`data.table`: Package showcasing a faster version of data.frame

Next Steps:

R Programming Reference: `http://rpubs.com/uwaterloodatateam/r-programming-reference`

Learn more R using swirl: `http://swirlstats.com/students.html`

Contribute useful code snippets: `https://github.com/uWaterlooDataTeam/r-programming-tutorial`