Introduction to R

Sarah Bonnin

2019-08-14

Contents

Chapter 1

Welcome

Dates, time & location

- Dates:
  - Module 1:
  - Module 2:
  - Module 3:
  - Module 4:
- Time:
  - 10:00-13:30
- Location:
  - CRG Training center

Instructors

Sarah Bonnin (Module 1, 2, 3) Julia Ponomarenko (Module 4) from the CRG Bioinformatics core facility (office 460, 4th floor hotel side)

Learning objectives

Chapter 2

What is R ?

- Programming language and environment for **data manipulation**, **statistical computing**, and **graphical display**.

- Implementation of the S programming language

- Created at the University of Auckland, New Zealand:

  - Initial version released in 1995
  - Stable version released in 2000

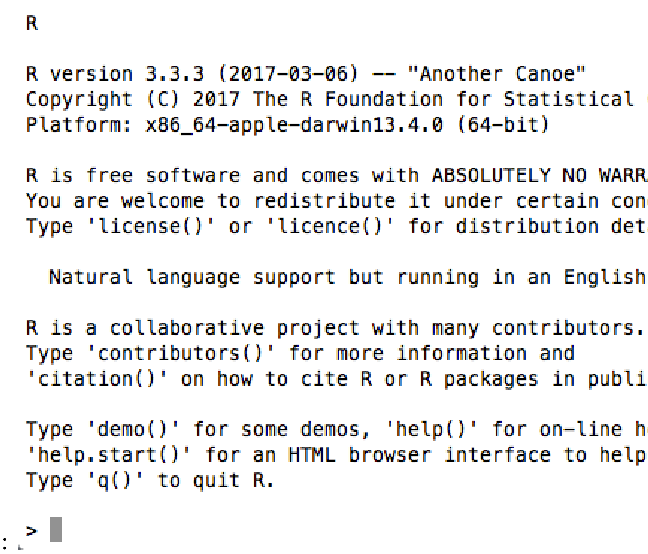
- **Free and open source !**

  - https://www.r-project.org/

- Interactive, flexible

- Very active community of developers and users!

  - Many resources and forums available

- Access through a command-line interpreter: 

```
R

R version 3.3.3 (2017-03-06) -- "Another Canoe"
Copyright (C) 2017 The R Foundation for Statistical
Platform: x86_64-apple-darwin13.4.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARR
You are welcome to redistribute it under certain con
Type 'license()' or 'licence()' for distribution det

  Natural language support but running in an English

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publi

Type 'demo()' for some demos, 'help()' for on-line h
'help.start()' for an HTML browser interface to help
Type 'q()' to quit R.

>
```

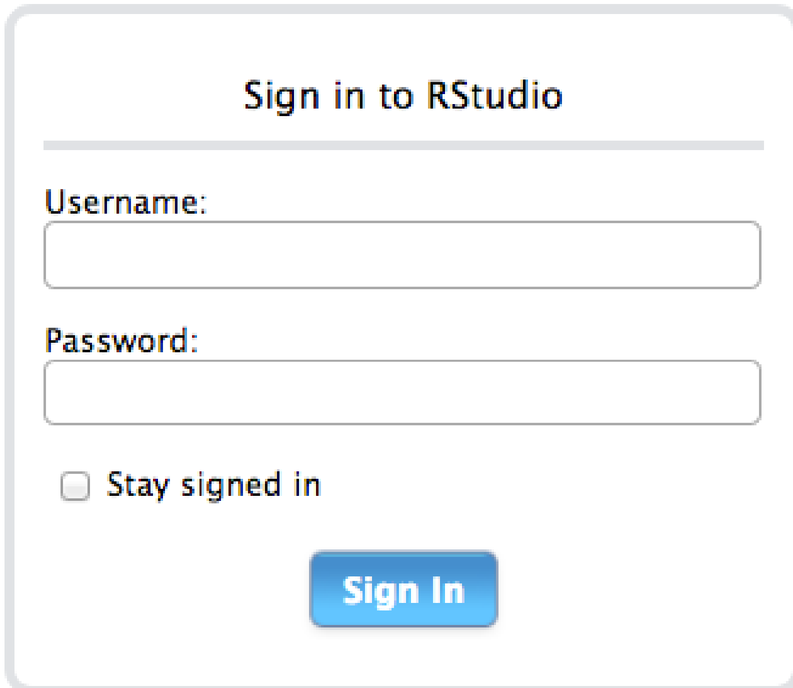
Chapter 3

What is RStudio ?

- Free and open source IDE (Integrated Development Environment) for R
- Available for Windows, Mac OS and LINUX

3.1 RStudio access

- RStudio Desktop installation
- RStudio access from the CRG server
  - Access with CRG credentials
  - For those who don't have access to the CRG server, use the guest accounts.

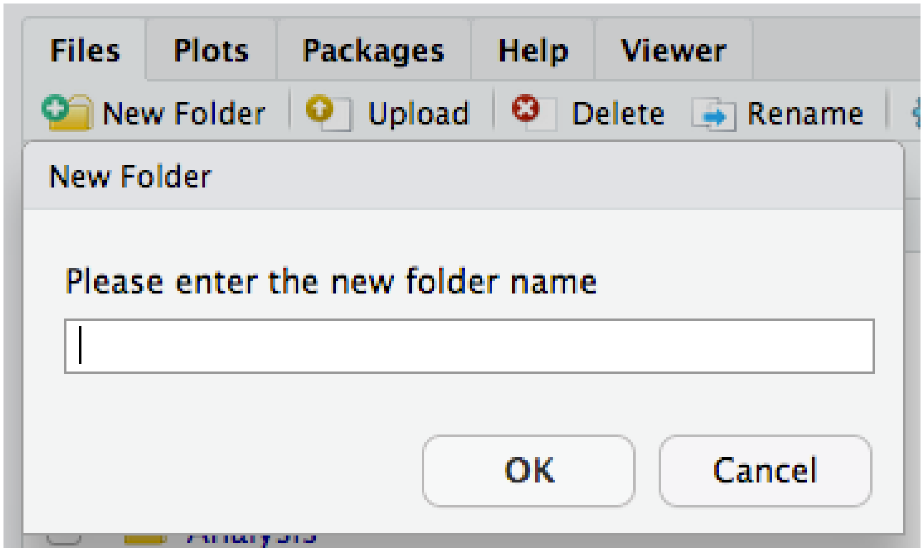


3.2 RStudio interface

- 4 panels:
  - top-left: scripts and files
  - bottom-left: R terminal
  - top-right: objects, history and environment
  - bottom-right: tree of folders, graph window, packages, help window, viewer

3.3 Setting up the folder structure for the course

Rcourse |-Module1 |-Module2 |-Module3 |-Module4


Files
Plots
Packages
Help
Viewer
New Folder
Upload
Delete
Rename
New Folder
Please enter the new folder name
OK
Cancel

Chapter 4

Paths and directories

- The path of a file/directory is its **location/address** in the file system.
- Your home directory is the one that hosts your personal folder:
  - for CRG users: **/nfs/users/[yourgroup]/[yourusername]**

4.1 Tree of directories

~: shortcut to the home directory .: current directory ..: one directory up the tree

4.2 Navigate the tree of directory with the R terminal

- Get the path of the current directory (know where you are working at the moment) with getwd (get working directory):

```
getwd()
```

- Change working directory with **setwd** (set working directory) Go to a directory giving the absolute path:

```
setwd("~/Rcourse")
```

Go to a directory giving the relative path:

```
setwd("Module1")
```

You are now in: "~/Rcourse/Module1" Move one directory "up" the tree:

```
setwd("..")
```

You are now in: "~/Rcourse"

Chapter 5

R basics

5.1 Arithmetic operators

Operator	Function
+	addition
-	subtraction
/	division
	multiplication
^ or **	exponential

In the R terminal:

```r
10 - 2
```

```
## [1] 8
```

Type **Enter** for R to interpret the command.

5.2 Simple calculations

Given the following table:

type of RNA	Total
mRNA	329
miRNA	45

type of RNA	Total
snoRNA	12
lncRNA	28

Calculate the total number of RNAs reported in the table:

```
329 + 45 + 12 + 28
```

```
## [1] 414
```

What is the percentage of miRNA?

```
( 45 / 414 ) * 100
```

```
## [1] 10.86957
```

5.3 Objects in R

Everything that stores any kind of data in R is an **object**:

#R syntax

5.4 Assignment operators

- **<-** or =
- Essentially the same but, to avoid confusions:
  - Use **<-** for assignments
  - Keep = for functions arguments

5.5 Assigning data to an object

- Assigning a value to the object **B**: B <- 10

- Reassigning: modifying the content of an object:

```
B + 10
```

B unchanged !!

```
B <- B + 10
```

B changed !!

- You can see the objects you created in the upper right panel in RStudio: the environment.

Chapter 6

Functions

In programming, a function is a section of a program that **performs a specific task**.

For example, the function **getwd** is used as:

```
getwd()
```

and has the task of outputting the current working directory.

You can recognize a function with the **round brackets**: function**()**

A function can also take *arguments/parameters*

```
setwd(dir="Rcourse")
```

setwd changes the current working directory and takes one argument **dir**.

- Assign the output of a function to an object:

- Getting help:

From the terminal:

```
help(getwd)
?getwd
```

From the RStudio bottom-right panel:

- The help pages show:

  - required/optional argument(s), if any.
  - default values for each argument(s), if any.
  - examples.
  - detailed description.

- Get the example of a function:

```
example(mean)
```

```
##
## mean> x <- c(0:10, 50)
##
## mean> xm <- mean(x)
##
## mean> c(xm, mean(x, trim = 0.10))
## [1] 8.75 5.50
```

- Need more help? Ask your favourite **Web search engine !**
- **Note on arguments**

The help page shows the compulsory arguments in the **Usage** section: in the help page of getwd and setwd (above), you can see that getwd doesn't take any compulsory argument, and setwd takes one compulsory argument that is called dir. Compulsory arguments can be given **with their names**: in such case you don't need to respect a specific order, or **without their names**, in which case you have to respect the order specified in the help page! For example, the **rep.int** function (a variant of the rep function) takes 2 arguments (see in help page): **x** and **times**, in that order:

```
# use arguments with their names:
rep.int(x=1, times=3)
```

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

```
# use arguments with their names without respecting the order:
rep.int(times=3, x=1)
```

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

```
# use arguments without their names but respecting the order:
rep.int(1, 3)
```

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

```
# use arguments without their names without respecting the order:
rep.int(3, 1)
```

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

```
# It works, but is not giving the expected output!
```

Chapter 7

R scripts

7.1 Create and save a script

- Store commands in a .R/.r script. Create and save a script in RStudio with:
  - File -> New File -> R Script
  - Once the file has opened: File -> Save
  - Specify a name: ***the extension .R is automatically added***
- Execute commands or blocks of commands from RStudio:

7.2 R syntax

- Case sensitive: **g** is not **G**
- Comment lines start with #
- Commands are separated by a **new line** or **;**

```
# This is a comment: it will not be interpreted
a <- 10
A + 1
# Will throw an error because A and a are different
```

7.3 RStudio tips in the console

Ctrl + Enter: execute the current line.

Upper arrow: goes to the commands previously typed. Ctrl + cmd + : Browse command history.

Type a letter in the console + "tab": R Studio proposes the different functions or object stored which start with that letter. for example, type **get + "tab"**:

7.4 Exercice 1. Getting started.

Create the script "exercise1.R" (in R Studio: File -> New File) and save it to the "Rcourse/Module1" directory: you will save all the commands of exercise 1 in that script. Remember you can comment the code using #.

1- From the terminal, go to Rcourse/Module1. First check where you currently are with getwd(); then go to Rcourse/Module1 with setwd()

correction

```
getwd()
setwd("Rcourse/Module1")
setwd("~/Rcourse/Module1")
```

2- Using R as a calculator, calculate the square root of 654.

correction

```
sqrt(654)
```

```
## [1] 25.57342
```

3- Using R as a calculator, calculate the percentage of males and females currently present in the classroom.

correction

```
# 6 males out of 19 students:
(6/19) * 100
```

```
## [1] 31.57895
```

```
# 13 females out of 19 students
(13/19) * 100
```

```
## [1] 68.42105
```

4- Create a new object "myobject" with value 60. Show "myobject" in the terminal.

correction

```
myobject <- 60
myobject
```

```
## [1] 60
```

5- Reassign myobject with value 87.

correction

```
myobject <- 87
```

6- Subtract 1 to myobject. Reassign.

correction

```
myobject <- myobject - 1
```

7- Create a new object "mysqrt" that will store the square root of "myobject".

correction

```
mysqrt <- sqrt(myobject)
```

8- Create a new object "mydiv" that will store the result of "myobject" divided by "mysqrt".

correction

```
mydiv <- myobject / mysqrt
```

Chapter 8

Data types

Each object has a data type: * Numeric (number - integer or double) * Character (text) * Logical (TRUE / FALSE)

##Checking data types

Number:

```
a <- 10
mode(a)
```

```
## [1] "numeric"
```

```
typeof(a)
```

```
## [1] "double"
```

```
str(a)
```

```
##  num 10
```

Text:

```
b <- "word"
mode(b)
```

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

```
typeof(b)
```

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

```
str(b)
```

```
##  chr "word"
```

Chapter 9

Data structures

The main data structures are:

- Vector
- Factor
- Matrix
- Data frame

9.1 Vectors

A vector is a sequence of data elements from the **same type**.

329 | 45 | 12 | 28 |

9.1.1 Creating a vector

- Values are assigned to a vector using the **c** command (**c**ombining elements).

```
a <- c(329, 45, 12, 28)
```

You can create an empty vector with:

```
vecempty <- vector()
```

- Create a sequence of consecutive numbers:

```
a <- 1:6
# same as:
a <- c(1, 2, 3, 4, 5, 6)
# both ends (1 and 6) are included
```

- Character vectors: Each element is entered between (single or double) quotes.

mRNA | miRNA | snoRNA | lncRNA |

```
b <- c("mRNA", "miRNA", "snoRNA", "lncRNA")
```

9.1.2 Vector manipulation

- A vector can be **named**: each element of the vector can be assigned a name (number or character)

```
names(a) <- c("mRNA", "miRNA", "snoRNA", "lncRNA")
# use an object which already contains a vector
names(a) <- b
```

- Get the length (number of elements) of a vector

```
length(a)
```

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

- Extracting elements from vector **a**
  - extract elements using their position (index) in the vector:

  ```
  a <- 1:6
  a[1]
  ```

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

  ```
  a[c(1,3)]
  ```

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

```
a[2:4]
```

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

- extract elements using their names:

```
a["mRNA"]
```

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

```
a[c("miRNA", "lncRNA")]
```

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

- Reassigning a vector's element

```
a[2] <- 31
a["miRNA"] <- 31
```

- Removing a vector's element

```
a <- a[-3]
```

- **Show** versus **change**

x[-2] x unchanged !

x <- x[-2] x reassigned !

9.1.3 Combining vectors

- From 2 vectors **a** and **b** you can create a vector **d**

```
a <- 2:5
b <- 4:6
d <- c(a, b)
```

The elements of **b** are added after the elements of **a**

- Likewise, you can add elements at the end of a vector

```
d <- c(d, 19)
```

9.1.4 Numeric vector manipulation

Logical operators

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	not x
x | y	x OR y
x & y	x AND y

- Which elements of **a** are equal to 2?

```
a <- 1:5
a == 2
```

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

- Which elements of **a** are superior to 2?

```
a <- 1:5
a > 2
```

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

- Extract elements of a vector that comply with a condition:

```
a <- 1:5
a >= 2
```

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

```
a[a >= 2]
```

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

9.1.4.1 Operations on vectors

- Adding 2 to a vector adds 2 to **each element** of the vector:

```
a <- 1:5
a + 2
```

```
## [1] 3 4 5 6 7
```

Same goes for subtractions, multiplications and divisions…

- Multiplying a vector by another vector of equal length

```
a <- c(2, 4, 6)
b <- c(2, 3, 0)
a * b
```

```
## [1]  4 12  0
```

- Multiplying a vector by another **shorter** vector

```
a <- c(2, 4, 6, 3, 1)
b <- c(2, 3, 0)
a * b
```

```
## Warning in a * b: longer object length is not a multiple of shorter object
## length
```

```
## [1]  4 12  0  6  3
```

Vector **a** is "recycled" !

- Summary statistics

Function	Description
mean(x)	mean / average
median(x)	median
min(x)	minimum
max(x)	maximum
var(x)	variance
summary(x)	mean, median, min, max, quartiles

```
a <- c(1, 3, 12, 45, 3, 2)
summary(a)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    2.25    3.00   11.00    9.75   45.00
```

9.1.4.2 Comparing vectors

- The **%in%** operator

Which elements of **a** are also found in **b* ?

```
a <- 2:6
b <- 4:10
a %in% b
```

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

Retrieve actual elements of **a** that are found in **b**:

```
a <- 2:6
b <- 4:10
a[a %in% b]
```

```
## [1] 4 5 6
```

9.1.5 Character vector manipulation

Character vectors are manipulated similarly to numeric ones.

- The **%in%** operator:

```
k <- c("mRNA", "miRNA", "snoRNA", "RNA", "lincRNA")
p <- c("mRNA","lincRNA", "tRNA", "miRNA")
k %in% p
```

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

```
k[k %in% p]
```

```
## [1] "mRNA"    "miRNA"   "lincRNA"
```

- Select elements from vector **m** that are not *exon*

```
m <- c("exon", "intron", "exon")
m != "exon"
```

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

```
m[m != "exon"]
```

```
## [1] "intron"
```

9.2 Exercise 2. Numeric vector manipulation

9.2.1 Exercise 2a.

Create the script “exercise2.R” and save it to the “Rcourse/Module1” directory: you will save all the commands of exercise 2 in that script. Remember you can comment the code using #.

1- Go to Rcourse/Module1 First check where you currently are with getwd(); then go to Rcourse/Module1 with setwd()

correction

```
getwd()
setwd("Rcourse/Module1")
setwd("~/Rcourse/Module1")
```

2- Create a numeric vector y which contains the numbers from 2 to 11, both included. Show y in the terminal.

correction

```r
y <- c(2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
# same as
y <- 2:11
# show in terminal:
y
```

```
##  [1]  2  3  4  5  6  7  8  9 10 11
```

3- How many elements are in y? I.e what is the length of vector y ?

correction

```r
length(y)
```

```
## [1] 10
```

4- Show the 2nd element of y.

correction

```r
y[2]
```

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

5- Show the 3rd and the 6th elements of y.

correction

```r
y[c(3,6)]
```

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

6- Remove the 4th element of y: reassign. What is now the length of y ?

correction

```r
# remove 4th element and reassign
y <- y[-4]
# length of y
length(y)
```

```
## [1] 9
```

7- Show all elements of y that are less than 7.

correction

```
# which elements of y are less than 7:
y < 7
```

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

```
# show those elements
y[ y < 7 ]
```

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

8- Show all elements of y that are greater or equal to 4 and less than 9.

correction

```
y[ y >= 4 & y < 9 ]
```

```
## [1] 4 6 7 8
```

9- Create the vector x of 1000 random numbers from the normal distribution: *First read the help page of the rnorm() function.*

correction

```
# help page for the rnorm function
help(rnorm)
# produce a vector of 1000 random numbers from the normal distribution
x <- rnorm(1000)
```

10. What are the mean, median, minimum and maximum values of x?

correction

```
mean(x); median(x); min(x); max(x)
```

```
## [1] -0.0024426
```

```
## [1] -0.01987436
```

```
## [1] -4.276704
```

```
## [1] 3.721429
```

11- Run the summary() function on x. What additional information do you obtain ?

correction

```
summary(x)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -4.277000 -0.689500 -0.019870 -0.002443  0.625900  3.721000
```

12- Create vector y2 as:

```
y2 <- c(1, 11, 5, 62,  18, 2, 8)
```

13. What is the sum of all elements in y2 ?

correction

```
sum(y2)
```

```
## [1] 107
```

14- Which elements of y2 are also present in y ? Note: remember the %in% operator.

correction

```
y2[ y2 %in% y ]
```

```
## [1] 11  2  8
```

15- Multiply each element of y2 by 1.5: reassign.

correction

```
y2 <- y2 * 1.5
```

16- Use the function any() to check if the number 3 is present.

correction

```
# "Given a set of logical vectors, is at least one of the values true?"
any( y2 == 3 )
```

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

9.2.2 Exercise 2b.

1- Create the vector myvector as:

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

Create the same vector using the rep() function (?rep)

correction

```
myvector <- rep(1:3, 3)
```

2- Reassign the 5th, 6th and 7th position of myvector with the values 8, 12 and 32, respectively.

correction

```
# reassign one by one
myvector[5] <- 8
myvector[6] <- 12
myvector[7] <- 32
# or reassign all at once
myvector[5:7] <- c(8, 12, 32)
```

3- Calculate the fraction/percentage of each element of myvector (relative to the sum of all elements of the vector). sum() can be useful.

correction

```
# sum of all elements of the vector
mytotal <- sum(myvector)
# divide each element by the sum
myvector / mytotal
```

```
## [1] 0.015625 0.031250 0.046875 0.015625 0.125000 0.187500 0.500000 0.031250
## [9] 0.046875
```

```
# multiply by 100 to get a percentage
(myvector / mytotal) * 100
```

```
## [1]  1.5625  3.1250  4.6875  1.5625 12.5000 18.7500 50.0000  3.1250  4.6875
```

4- Add vector c(2, 4, 6, 7) to myvector (combining both vectors): reassign!

correction

```
# create the new vector
newvector <- c(2, 4, 6, 7)
# combine both myvector and newvector
c(myvector, newvector)
```

```
##  [1]  1  2  3  1  8 12 32  2  3  2  4  6  7
```

```
# reassign myvector
myvector <- c(myvector, newvector)
```

9.3 Exercise 3. Character vector manipulation

9.3.1 Exercise 3a.

Create the script "exercise3.R" and save it to the "Rcourse/Module1" directory: you will save all the commands of exercise 3 in that script. Remember you can comment the code using #.

1- Go to Rcourse/Module1 First check where you currently are with getwd(); then go to Rcourse/Module1 with setwd()

correction

```
getwd()
setwd("Rcourse/Module1")
setwd("~/Rcourse/Module1")
```

2- Create vector w as:

```
w <- rep(x=c("miRNA", "mRNA"), times=c(3, 2))
```

3- View vector w in the console: how does function rep() work ? Play with the **times** argument.

correction

```r
rep(x=c("miRNA", "mRNA"), times=c(3, 4))
```

```
## [1] "miRNA" "miRNA" "miRNA" "mRNA"  "mRNA"  "mRNA"  "mRNA"
```

```r
rep(x=c("miRNA", "mRNA"), times=c(10, 2))
```

```
##  [1] "miRNA" "miRNA" "miRNA" "miRNA" "miRNA" "miRNA" "miRNA" "miRNA"
##  [9] "miRNA" "miRNA" "mRNA"  "mRNA"
```

4- What is the output of table(w) ? What does the table function do ?

5- Type w[grep(pattern="mRNA", x=w)] and w[w == "mRNA"] Is there a difference between the two outputs?

correction

```r
w[grep(pattern="mRNA", w)]
```

```
## [1] "mRNA" "mRNA"
```

```r
w[w == "mRNA"]
```

```
## [1] "mRNA" "mRNA"
```

```r
# no difference between the outputs
```

6- Now type w[grep(pattern="RNA", w)] and w[w == "RNA"] Is there a difference between the two outputs?

correction

```r
w[grep(pattern="RNA", w)]
```

```
## [1] "miRNA" "miRNA" "miRNA" "mRNA"  "mRNA"
```

```r
w[w == "RNA"]
```

```
## character(0)
```

```
# grep outputs 5 values but == outputs none
```

What is the difference between == and **grep** ?

correction

> = looks for exact matches. grep looks for **patterns**.

7- Create vector g as:

```
g <- c("hsa-let-7a", "hsa-mir-1", "CLC", "DKK1", "LPA")
```

How many elements do w and g contain?

correction

```
length(w); length(g)
```

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

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

8- Do vectors w and g have the same length? Use the function identical() to check this.

correction

```
identical(x=length(w), y=length(g))
```

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

9- Name the elements of g using the elements of w. (i.e. the names of each element of g will be the elements of w).

correction

```
names(g) <- w
```

If you have time, continue with Exercise 3b below.

9.3.2 Exercise 3b.

1- Use the sub() function to replace miRNA with microRNA in the *names* of g.

correction

```
names(g) <- sub(pattern="miRNA", replacement="microRNA", x=names(g))
```

2- Count how many microRNAs and mRNAs there are in g based on the column names.

correction

```
table(names(g))
```

```
##
## microRNA     mRNA
##        3        2
```

3- Create vector tt as:

```
tt <- "Introduction to R course"
```

How many characters does tt contain? Use nchar().

correction

```
nchar(tt)
```

```
## [1] 24
```

4- Remove "Introduction to R" from tt. You can try with either substr() or gsub()

correction

```
substr(x=tt, start=17, stop=nchar(tt))
```

```
## [1] "R course"
```

```
gsub(pattern="Introduction to R", replacement="", x=tt)
```

```
## [1] " course"
```

9.4 Factors

- A factor is a vector object (1 dimension) used to specify a **discrete classification (grouping)** of the components of other vectors.
- Factors are mainly used for **statistical modeling**, and can also be useful for graphing.
- You can create factors with the **factor** function, for example:

```
e <- factor(c("high", "low", "medium", "low"))
# check the structure of e
str(e)
```

```
##  Factor w/ 3 levels "high","low","medium": 1 2 3 2
```

- Example of a character vector versus a factor

```
# factor
e <- factor(c("high", "low", "medium", "low"))
# character vector
e2 <- c("high", "low", "medium", "low")
# Check the structure of both objects
str(e)
```

```
##  Factor w/ 3 levels "high","low","medium": 1 2 3 2
```

```
str(e2)
```

```
##  chr [1:4] "high" "low" "medium" "low"
```

- Groups in factors are called **levels**. Levels can be **ordered**. Then, some operations applied on numeric vectors can be used:

```
# unordered factor:
e <- factor(c("high", "low", "medium", "low"))
max(e) # throws an error
# ordered factor
e_ord <- factor(e, levels=c("low", "medium", "high"), ordered=TRUE)
max(e_ord) # outputs "high"
```

9.5 Matrices

- A matrix is a **2 dimensional** vector.
- All columns in a matrix must have:
  - the same **type** (numeric, character or logical)
  - the same **length**

9.5.1 Creating a matrix

- From vectors with the **rbind** function:

```
x <- c(1, 44)
y <- c(0, 12)
z <- c(34, 4)
# rbind: bind rows
b <- rbind(x, y, z)
```

- From vectors with the **cbind** function:

```
i <- c(1, 0, 34)
j <- c(44, 12, 4)
# cbind: bind columns
b <- cbind(i, j)
```

- From scratch with the *matrix* function:

```
# nrow: number of rows
# ncol: number of columns
b <- matrix(c(1, 0, 34, 44, 12, 4),
    nrow=3,
    ncol=2)
```

9.5.2 Two-dimensional object

Vectors have one index per element (1-dimension). Matrices have **two indices (2-dimensions)** per element, corresponding to the row and the column:

- Fetching elements of a matrix:

The "coordinates" of an element in a 2-dimensional object will be first the row (on the left of the comma), then the column (on the right of the comma):

9.5.3 Matrix manipulation

- Add 1 to all elements of a matrix

```
b <- b + 1
```

- Multiply by 3 all elements of a matrix

```
b <- b * 3
```

- Subtract 2 to each element of **the first row** of a matrix

```
b[1, ] <- b[1, ] - 2
```

- Replace elements that comply a condition:

```
# Replace all elements that are greater than 3 with NA
b[ b>3 ] <- NA
```

9.6 Data frames

A data frame is a 2-dimensional structure. It is more general than a matrix. All columns in a data frame: + can be of different **types** (numeric, character or logical) + must have the same **length**

9.6.1 Create a data frame

- With the **data.frame** function:

```
# stringsAsFactors: ensures that characters are treated as characters and not as facto
d <- data.frame(c("Maria", "Juan", "Alba"),
    c(23, 25, 31),
    c(TRUE, TRUE, FALSE),
    stringsAsFactors = FALSE)
```

- Example why “stringsAsFactors = FALSE” is useful

```
# Create a data frame with default parameters
df <- data.frame(label=rep("test",5), column2=1:5)
# Replace one value
df[2,1] <- "yes"
```

```
## Warning in `[<-.factor`(`*tmp*`, iseq, value = "yes"): invalid factor
## level, NA generated
```

```r
# Throws an error and doesn't replace the value !
```

```r
# Create a data frame with
df2 <- data.frame(label=rep("test",5), column2=1:5, stringsAsFactors = FALSE)
# Replace one value
df2[2,1] <- "yes"
# Works!
```

- Converting a matrix into a data frame:

```r
# create a matrix
b <- matrix(c(1, 0, 34, 44, 12, 4),
        nrow=3,
        ncol=2)
# convert as data frame
b_df <- as.data.frame(b)
```

9.6.2 Data frame manipulation:

Very similar to matrix manipulation.

9.7 Two-dimensional structures manipulation

9.7.1 Dimensions

- Get the number of rows and the number of columns:

```r
# Create a data frame
d <- data.frame(c("Maria", "Juan", "Alba"),
    c(23, 25, 31),
    c(TRUE, TRUE, FALSE),
    stringsAsFactors = FALSE)
# number of rows
nrow(d)
```

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

```
# number of columns
ncol(d)
```

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

- Check the dimensions of the object: both number of rows and number of columns:

```
# first element: number of rows
# second element: number of columns
dim(d)
```

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

- Dimension names

Column and/or row names can be added to matrices and data frames

```
colnames(d) <- c("Name", "Age", "Vegetarian")
rownames(d) <- c("Patient1", "Patient2", "Patient3")
```

Column and/or row names can be used to retrieve elements or sets of elements from a 2-dimensional object:

```
d[,"Name"]
```

```
## [1] "Maria" "Juan"  "Alba"
```

```
# same as:
d[,1]
```

```
## [1] "Maria" "Juan"  "Alba"
```

```
d["Patient3", "Age"]
```

```
## [1] 31
```

```
# same as:
d[3,2]
```

```
## [1] 31
```

```
# for data frames only, the $ sign can be used to retrieve columns:
# d$Name is d[,1] is d[, "Name"]
```

- Include names as you create objects:
  - Matrix:

```
m <- matrix(1:4, ncol=2,
    dimnames=list(c("row1", "row2"), c("col1", "col2")))
```

```
+ Data frame:
```

```
df <- data.frame(col1=1:2, col2=1:2,
    row.names=c("row1", "row2"))
```

9.7.2 Manipulation

Same principle as vectors... but in 2 dimensions!

Examples

- select the columns of b if **at least one element in the 3rd row is less than or equal to 4**:

```
# create b
b <- matrix(c(1, 0, 34, 44, 12, 4),
    nrow=3,
    ncol=2)
# third row of b:
b[3, ]
```

```
## [1] 34  4
```

```
# element(s) in the third row of b that is (are) less than or equal to 4
b[3, ] <= 4
```

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

```
# retrieve the corresponding sub-matrix
b[ ,b[3, ] <= 4]
```

```
## [1] 44 12  4
```

- Select rows of b if **at least one element in column 2 is greater than 24**:

```
# build data frame d
d <- data.frame(Name=c("Maria", "Juan", "Alba"),
        Age=c(23, 25, 31),
        Vegetarian=c(TRUE, TRUE, FALSE),
        stringsAsFactors = FALSE)
rownames(d) <- c("Patient1", "Patient2", "Patient3")
# The following commands all output the same result:
d[d[,2] > 24, ]
```

```
##          Name Age Vegetarian
## Patient2 Juan  25       TRUE
## Patient3 Alba  31      FALSE
```

```
d[d[,"Age"] > 24, ]
```

```
##          Name Age Vegetarian
## Patient2 Juan  25       TRUE
## Patient3 Alba  31      FALSE
```

```
d[d$Age > 24, ]
```

```
##          Name Age Vegetarian
## Patient2 Juan  25       TRUE
## Patient3 Alba  31      FALSE
```

- Select patients (rows) based on 2 criteria: age of the patient (column 2) should be great than or equal to 25, and the patient should be vegetarian (column 3):

```
d[ d$Age >= 25 & d$Vegetarian == TRUE, ]
```

```
##          Name Age Vegetarian
## Patient2 Juan  25       TRUE
```

More useful commands

- Add a row or a column with **rbind** and **cbind**, respectively

```
# add a column
cbind(d, 1:3)
```

```
##           Name Age Vegetarian 1:3
## Patient1 Maria  23       TRUE   1
## Patient2  Juan  25       TRUE   2
## Patient3  Alba  31      FALSE   3
```

```
# add a row
rbind(d, 4:6)
```

```
##           Name Age Vegetarian
## Patient1 Maria  23          1
## Patient2  Juan  25          1
## Patient3  Alba  31          0
## 4            4   5          6
```

Add a patient to our data frame **d**:

```
d <- rbind(d, c("Jordi", 33, FALSE))
```

- Process the sum of all rows or all columns with **rowSums** and **colSums**, respectively.

```
# create a matrix
b <- matrix(1:20, ncol=4)
# process sum of rows and sum of cols
rowSums(b)
```

```
## [1] 34 38 42 46 50
```

```
colSums(b)
```

```
## [1] 15 40 65 90
```

- The **apply** function

Powerful tool to apply a command to all rows or all columns of a data frame or a matrix. For example, instead of calculating the sum of each row, you might be interested in calculating the median ? But rowMedians doesn't exist ! **apply** takes 3 arguments: - first argument **X**: 2-dimensional object - second argument **MARGIN**: apply by row or by column? + 1: by row + 2: by column - third argument **FUN**: function to apply to either rows or columns

```
# median value of each row of b
apply(X=b, MARGIN=1, FUN=median)
```

```
## [1]  8.5  9.5 10.5 11.5 12.5
```

```
# median value of each column of b
apply(X=b, MARGIN=2, FUN=median)
```

```
## [1]  3  8 13 18
```

9.8 Exercise 4. Matrix manipulation

Create the script “exercise4.R” and save it to the “Rcourse/Module1” directory: you will save all the commands of exercise 4 in that script. Remember you can comment the code using #.

correction

```
getwd()
setwd("Rcourse/Module1")
setwd("~/Rcourse/Module1")
```

1- Create three numeric vectors x, y, z, each of 4 elements of your choice.

correction

```
x <- 2:5
y <- 6:9
z <- 7:4
```

Use rbind() to create a matrix **mat** (3 rows and 4 columns) out of x, y and z.

correction

```
mat <- rbind(x, y, z)
```

2- Create the same matrix now using the matrix function.

correction

```
mat <- matrix(data=c(x, y, z), nrow=3, ncol=4)
# Try with the "byrow=TRUE" parameter: what is different ?
mat <- matrix(data=c(x, y, z), nrow=3, ncol=4, byrow=TRUE)
```

3- Add names to mat's columns: "a", "b", "c", "d", respectively.

correction

```
colnames(mat) <- c("a", "b", "c", "d")
```

4- Calculate the sum of each row, and the sum of each column

correction

```
rowSums(mat); colSums(mat)
```

```
## [1] 14 30 22
```

```
##  a  b  c  d
## 15 16 17 18
```

5- Create the matrix mat2 as:

```
mat2 <- matrix(c(seq(from=1, to=10, by=2), 5:1, rep(x=2017, times=5)), ncol=3)
```

What does function seq() do?

correction

seq generate sequences of numbers. Here, it creates a sequences from 1 to 10 with a step of 2 numbers.

6- What are the dimensions of mat2 (number of rows and number of columns)?

correction

```
# number of rows
nrow(mat2)
```

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

```
# number of columns
ncol(mat2)
```

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

```
# dimensions: number of rows, number of columns
dim(mat2)
```

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

7- Add column names to mat2: “day”, “month” and “year”, respectively.

correction

```
colnames(mat2) <- c("day", "month", "year")
```

8- Add row names to mat2: letters “A” to “E”

correction

```
rownames(mat2) <- c("A", "B", "C", "D", "E")
rownames(mat2) <- LETTERS[1:5]
```

9- Shows row(s) of mat2 where the month column is greater than or equal to 3.

correction

```
# select column month
mat2[, "month"]
```

```
## A B C D E
## 5 4 3 2 1
```

```
# element(s) of column month that is (are) greater than or equal to 3
mat2[,"month"] >= 3
```

```
##     A     B     C     D     E
##  TRUE  TRUE  TRUE FALSE FALSE
```

```
# finally select row(s) where the month columns is greater than or equal to 3
mat2[mat2[,"month"] >= 3,]
```

```
##   day month year
## A   1     5 2017
## B   3     4 2017
## C   5     3 2017
```

10- Replace all elements of mat2 that are equal to 2017 with 2018.

correction

```
# which elements of mat2 that are exactly equal to 2017
mat2==2017
```

```
##     day month year
## A FALSE FALSE TRUE
## B FALSE FALSE TRUE
## C FALSE FALSE TRUE
## D FALSE FALSE TRUE
## E FALSE FALSE TRUE
```

```
# retrieve actual elements
mat2[mat2==2017]
```

```
## [1] 2017 2017 2017 2017 2017
```

```
# replace all 2017 with 2018
mat2[mat2==2017] <- 2018
```

11- Multiply all elements of the 2nd column of mat2 by 7. Reassign mat2!

correction

```
# multiply all elements of the 2nd column of mat2 by 7
mat2[,2] * 7
```

```
##  A  B  C  D  E
## 35 28 21 14  7
```

```
# reassign mat2 with the new values of column 2
mat2[,2] <- mat2[,2] * 7
```

12- Add the column named "time" to mat2, that contains values 8, 12, 11, 10, 8. Save in the new object mat3.

correction

```
mat3 <- cbind(mat2, time=c(8, 12, 11, 10, 8))
```

13- Replace all elements of mat3 that are less than 3 with NA.

correction

```r
# which elements of mat3 that are less than 3
mat3 < 3
```

```
##     day month  year  time
## A  TRUE FALSE FALSE FALSE
## B FALSE FALSE FALSE FALSE
## C FALSE FALSE FALSE FALSE
## D FALSE FALSE FALSE FALSE
## E FALSE FALSE FALSE FALSE
```

```r
# actually elements of mat3 that are less than 3
mat3[mat3 < 3]
```

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

```r
# reassign elements of mat3 that are less than 3 with NA
mat3[mat3 < 3] <- NA
```

14- Remove rows from mat3 if a NA is present. Save in the new object mat4.

correction

```r
mat4 <- na.omit(mat3)
```

15- Retrieve the smaller value of each column of mat4.

Try different approaches:

- Retrieve the minimum for each column one by one.

correction

```r
min(mat4[,"day"])
```

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

```r
min(mat4[,"month"])
```

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

```
min(mat4[,"year"])
```

```
## [1] 2018
```

```
min(mat4[,"time"])
```

```
## [1] 8
```

- Retrieve the minimum of all columns simultaneously using the apply() function.

correction

```
# mat4: object
# 2: by column
# min: function to apply
apply(mat4, 2, min)
```

```
##   day month  year  time
##     3     7  2018     8
```

9.9 Exercise 5. Data frame manipulation

Create the script "exercise5.R" and save it to the "Rcourse/Module1" directory: you will save all the commands of exercise 5 in that script. Remember you can comment the code using #.

correction

```
getwd()
setwd("Rcourse/Module1")
setwd("~/Rcourse/Module1")
```

9.9.1 Exercise 5a

1- Create the following data frame:

|43|181|M| |34|172|F| |22|189|M| |27|167|F|

With Row names: John, Jessica, Steve, Rachel. And Column names: Age, Height, Sex.

correction

```
df <- data.frame(Age=c(43, 34, 22, 27),
                 Height=c(181, 172, 189, 167),
                 Sex=c("M", "F", "M", "F"),
                 row.names = c("John", "Jessica", "Steve", "Rachel"),
                 stringsAsFactors=FALSE)
```

2- Check the structure of df with str().

correction

```
str(df)
```

3- Calculate the average age and height in df

Try different approaches: * Calculate the average for each column separately.

correction

```
mean(df$Age)
mean(df$Height)
```

- Calculate the average of both columns simultaneously using the apply() function.

correction

```
# we have to remove the Sex column: we can calculate the average only with numbers
apply(df[,-3], 2, mean)
apply(df[,1:2], 2, mean)
apply(df[,-grep("Sex", colnames(df))], 2, mean)
```

4- Add one row to df2: Georges who is 53 years old and 168 tall.

correction

```
# Georges= allows us to enter the row name at the same time as we add a row
df <- rbind(df, Georges=c(53, 168, "M"))
```

5- Change the row names of df so the data becomes anonymous: Use Patient1, Patient2, etc. instead of actual names.

correction

```
rownames(df) <- c("Patient1", "Patient2", "Patient3", "Patient4", "Patient5")
# try also the paste function!
rownames(df) <- paste("Patient", 1:5, sep="")
```

6- Create the data frame df2 that is a subset of df which will contain only the female entries.

correction

```
# which elements are female ("F" in the "Sex" colum)
df$Sex=="F"
# retrieve rows that contain the female entries, and save in df2
df2 <- df[df$Sex=="F",]
```

7- Create the data frame df3 that is a subset of df which will contain only entries of males taller than 170.

correction

```
# which entries are males
df$Sex=="M"
# which entries are greater than 170 in column "Height"
df$Sex=="M" & df$Height > 170
# retrieve rows that contain the males that are taller than 170, and save in df3
df3 <- df[df$Sex=="M" & df$Height > 170,]
```

9.9.2 Exercise 5b

1. Create two data frames mydf1 and mydf2 as:

mydf1:

|1|14| |2|12| |3|15| |4|10|

mydf2:

|1|paul| |2|helen| |3|emily| |4|john| |5|mark|

With column names: **"id", "age"** for mydf1, and **"id", "name"** for mydf2.

correction

```
mydf1 <- data.frame(id=1:4, age=c(14,12,15,10))
mydf2 <- data.frame(id=1:5, name=c("paul", "helen", "emily", "john", "mark"))
```

2- Merge mydf1 and mydf2 by their "id" column. Look for the help page of **merge** and/or Google it!

correction

```
# input 2 data frames
# "by" columns indicate by which column you want to merge the data
merge(x=mydf1, y=mydf2, by.x="id", by.y="id")
mydf3 <- merge(x=mydf1, y=mydf2, by="id")
```

3- Order mydf3 by decreasing age. Look for the help page of **order**.

correction

```
# order the age column (default is increasing order)
order(mydf3$age)
# order the age column by decreasing order
order(mydf3$age, decreasing = TRUE)
# order the whole data frame by the column age in decreasing order
mydf3[order(mydf3$age, decreasing = TRUE), ]
```

9.9.3 Exercise 5c

1- Using the download.file function, download this file to your current directory. (Right click on “this file” -> Copy link location to get the full path).

correction

```
# failing: download.file("https://github.com/sbcrg/CRG_RIntroduction/blob/master/genes_
download.file("https://public-docs.crg.es/biocore/sbonnin/Rcourse/genes_dataframe.RData
```

2- The function dir() lists the files and directories present in the current directory: check if genes_dataframe.RData was copied.

correction

```
dir()
```

3- Load genes_dataframe.RData in your environment Use the *load* function.

correction

```
load("genes_dataframe.RData")
```

4- genes_dataframe.RData contains the df_genes object: is it now present in your environment?

correction

```
ls()
```

5- Explore df_genes and see what it contains You can use a variety of functions: str, head, tail, dim, colnames, rownames, class…

correction

```
str(df_genes)
head(df_genes)
tail(df_genes)
dim(df_genes)
colnames(df_genes)
rownames(df_genes)
class(df_genes)
```

6- Select rows for which pvalue_KOvsWT $<$ 0.05 AND log2FoldChange_KOvsWT $>$ 0.5. Store in the up object.

correction

```
# rows where pvalue_KOvsWT < 0.05
df_genes$pvalue_KOvsWT < 0.05
# rows where log2FoldChange_KOvsWT > 0.5
df_genes$log2FoldChange_KOvsWT > 0.5
# rows that comply both of the above conditions
df_genes$pvalue_KOvsWT < 0.05 & df_genes$log2FoldChange_KOvsWT > 0.5
# select rows for which pvalue_KOvsWT < 0.05 AND log2FoldChange_KOvsWT > 0.5
up <- df_genes[df_genes$pvalue_KOvsWT < 0.05 &
                 df_genes$log2FoldChange_KOvsWT > 0.5,]
```

How many rows (genes) were selected?

7- Select from the up object the Zinc finger protein coding genes (i.e. the gene symbol starts with Zfp). Use the grep() function.

correction

```
# extract gene symbol column
up$gene_symbol
# use grep to get the genes matching the pattern "Zfp"
up[grep("Zf", up$gene_symbol), ]
```

8- Select rows for which pvalue_KOvsWT $<$ 0.05 AND log2FoldChange_KOvsWT is $>$ 0.5 OR $<$ -0.5. For the selection of log2FoldChange: give the **abs** function a try! Store in the diff_genes object.

correction

```r
# rows where pvalue_KOvsWT < 0.05
df_genes$pvalue_KOvsWT < 0.05
# rows where log2FoldChange_KOvsWT > 0.5
df_genes$log2FoldChange_KOvsWT > 0.5
# rows where log2FoldChange_KOvsWT < -0.5
df_genes$log2FoldChange_KOvsWT > -0.5
# rows where log2FoldChange_KOvsWT < -0.5 OR log2FoldChange_KOvsWT > 0.5
df_genes$log2FoldChange_KOvsWT > 0.5 | df_genes$log2FoldChange_KOvsWT > -0.5
# same as above but using the abs function
abs(df_genes$log2FoldChange_KOvsWT) > 0.5
# combine all required criteria
df_genes$pvalue_KOvsWT < 0.05 & abs(df_genes$log2FoldChange_KOvsWT) > 0.5
# extract corresponding entries
diff_genes <- df_genes[df_genes$pvalue_KOvsWT < 0.05 &
                 abs(df_genes$log2FoldChange_KOvsWT) > 0.5,]
```

How many rows (genes) were selected?

Chapter 10

Missing values

NA (Not Available) is a recognized element in R.

- Finding missing values in a vector

```
# Create vector
x <- c(4, 2, 7, NA)

# Find missing values in vector:
is.na(x)

# Remove missing values
na.omit(x)
x[ !is.na(x) ]
```

- Some functions can deal with NAs, either by default, or with specific arguments:

```
x <- c(4, 2, 7, NA)

# default arguments
mean(x)

# set na.rm=TRUE
mean(x, na.rm=TRUE)
```

- In a matrix or a data frame, keep only rows where there are no NA values:

```
# Create matrix with some NA values
mydata <- matrix(c(1:10, NA, 12:2, NA, 15:20, NA), ncol=3)

# Keep only rows without NAs
mydata[complete.cases(mydata), ]
# or
na.omit(mydata)
```

Check this R blogger post on missing/null values

Chapter 11

Input / Output

We will learn how to: * Read in a file * Write out a file * Save a graph in a file (Module 3)

11.1 On vectors

- Read a file as a vector with the **scan** function

```
# Read in file
scan(file="file.txt")
# Save in  object
k <- scan(file="file.txt")
```

By default, scans "double" (numeric) elements: it fails if the input contains characters. If non-numeric, you need to specify the type of data contained in the file:

```
# specify the type of data to scan
scan(file="file.txt",
    what="character")
scan(file="~/file.txt",
    what="character")
```

Regarding paths of files: If the file is not in the current directory, you can provide a full or relative path. For example, if located in the home directory, read it as:

```
scan(file="~/file.txt",
    what="character")
```

- Write the content of a vector in a file:

```
# create a vector
mygenes <- c("SMAD4", "DKK1", "ASXL3", "ERG", "CKLF", "TIAM1", "VHL", "BTD", "EMP1", "
# write in a file
write(x=mygenes,
    file="gene_list.txt")
```

Regarding paths of files: When you write a file, you can also specify a full or relative path:

```
# Write to home directory
write(x=mygenes,
        file="~/gene_list.txt")
# Write to one directory up
write(x=mygenes,
        file="../gene_list.txt")
```

11.2 On data frames or matrices

- Read in a file into a data frame with the **read.table** function:

```
a <- read.table(file="file.txt")
```

You can convert it as a matrix, if needed, with:

```
a <- as.matrix(read.table(file="file.txt"))
```

Useful arguments:

- Write a data frame or matrix to a file:

```
write.table(x=a,
    file="file.txt")
```

Useful arguments:

- Note that ”̂” stands for tab-delimitation

11.3 Exercise 6.

Create the script "exercise6.R" and save it to the "Rcourse/Module2" directory: you will save all the commands of exercise 6 in that script. Remember you can comment the code using #.

correction

```
getwd()
setwd("Rcourse/Module2")
setwd("~/Rcourse/Module2")
```

11.3.1 Exercise 6a. Input / output

1- Download folder "i_o_files" in your current directory with:

```
# system invokes the OS command specified by the "command" argument.
system(command="svn export https://github.com/sarahbonnin/CRG_RIntroduction/trunk/i_o_files")
```

All files that will be used for exercise 6 are found in the **i_o_files** folder !

2- Read in the content of ex6a_input.txt using the scan command; save in object z

How many elements are in z?

correction

```
# scan content of the file
z <- scan("i_o_files/ex6a_input.txt")
# number of elements (length of vector)
length(z)
```

3- Sort z: save sorted vector in object "zsorted".

correction

```
zsorted <- sort(z)
```

4- Write zsorted content into file ex6a_output.txt.

correction

```
write(zsorted, "ex6a_output.txt")
```

5- Check the file you produced in the RStudio file browser (click on the file in bottom-right panel "Files" tab). Save the content of zsorted again but this time setting the argument "ncolumns" to 1: how is the file different?

correction

```
write(zsorted, "ex6a_output.txt", ncolumns=1)
```

11.3.2 Exercise 6b - I/O on data frame: play with the arguments of read.table

1- field separator

- Read ex6b_IO_commas_noheader.txt in object fs. What are the dimensions of fs?

correction

```
# read in file with default parameters
fs <- read.table("i_o_files/ex6b_IO_commas_noheader.txt")
dim(fs)
```

- Fields/columns are separated by commas: change the default value of the "sep" argument and read in the file again. What are now the dimensions of fs?

correction

```
# change field separator to ","
fs <- read.table("i_o_files/ex6b_IO_commas_noheader.txt",
    sep=",")
dim(fs)
```

2- field separator + header

- Read ex6b_IO_commas_header.txt in object fs_c. What are the dimensions of fs_c ?

correction

```
fs_c <- read.table("i_o_files/ex6b_IO_commas_header.txt")
dim(fs_c)
```

- Check head(fs_c) and change the default field separator to an appropriate one.

correction

```
fs_c <- read.table("i_o_files/ex6b_IO_commas_header.txt",
                   sep=",")
```

- The first row should to be the header (column names): change the default value of the header parameter and read in the file again. What are now the dimensions of fs_c ?

correction

```
fs_c <- read.table("i_o_files/ex6b_IO_commas_header.txt",
                   sep=",",
                   header=TRUE)
```

3- skipping lines

- Read ex6b_IO_skip.txt in object sk.

correction

```
sk <- read.table("i_o_files/ex6b_IO_skip.txt")
```

Is R complaining ?

Check "manually" the file (in the R Studio file browser).

- The skip argument allows you to ignore one or more line(s) before reading in a file. Introduce this argument with the appropriate number of lines to skip, and read the file again.

correction

```
sk <- read.table("i_o_files/ex6b_IO_skip.txt",
                 skip=2)
dim(sk)
```

- Is R still complaining? What are now the dimensions of sk ?
- Change the default field separator. What are now the dimensions of sk ?

correction

```
sk <- read.table("i_o_files/ex6b_IO_skip.txt",
                 skip=2,
                 sep=",",
                 header=T)
```

4- Comment lines

- Read ex6b_IO_comment.txt in object cl.

correction

```
cl <- read.table("i_o_files/ex6b_IO_comment.txt")
```

Is R complaining again ? Check manually the file and try to find out what is wrong…

What os the comment.char argument used for ? Adjust the comment.char argument and read the file again.

correction

```
cl <- read.table("i_o_files/ex6b_IO_comment.txt",
                 comment.char = "*")
```

- Adjust also the header and sep arguments to read in the file correctly. What are now the dimensions of cl?

correction

```
cl <- read.table("i_o_files/ex6b_IO_comment.txt",
                 comment.char = "*",
                 sep=",",
                 header=TRUE)
dim(cl)
```

4- final

- Read ex6b_IO_final.txt in object fin.

correction

```
fin <- read.table("i_o_files/ex6b_IO_final.txt")
```

- Adjust the appropriate parameters according to what you have learnt, in order to obtain the data frame "fin" of dimensions 167 x 4.

correction

```
fin <- read.table("i_o_files/ex6b_IO_final.txt",
                  sep=",",
                  header=TRUE,
                  skip=3,
                  comment.char="&"
                  )
```

11.3.3 Exercice 6c - I/O on a data frame

1- Read in file ex6c_input.txt in ex6 object

Warning: the file has a header ! Check the structure of ex6 (remember the **str** command).

correction

```
ex6 <- read.table("i_o_files/ex6c_input.txt",
                  header=TRUE)
str(ex6)
```

2- Now read in the same file but, this time, set the argument as.is to TRUE.

Check again the structure: what has changed ?

correction

```
ex6 <- read.table("i_o_files/ex6c_input.txt",
                  header=TRUE,
                  as.is=TRUE)
str(ex6)
```

3- What are the column names of ex6 ?

correction

```
colnames(ex6)
```

4- Change the name of the first column of ex6 from "State" to "Country".

correction

```
# extract all column names of ex6
colnames(ex6)
# extract the name of the first column only
colnames(ex6)[1]
# reassign name of the first column only
colnames(ex6)[1] <- "Country"
```

5- How many countries are in the Eurozone, according to ex6 ?

Remember the table function.

correction

```
table(ex6$Eurozone)
```

6- In the Eurozone column: change "TRUE" with "yes" and "FALSE" with "no".

correction

```
# select the Eurozone column
ex6$Eurozone
# elements of the Eurozone column that are exactly TRUE
ex6$Eurozone==TRUE
# extract actual values that are TRUE
ex6$Eurozone[ex6$Eurozone==TRUE]
# reassign all elements that are TRUE with "yes"
ex6$Eurozone[ex6$Eurozone==TRUE] <- "yes"
# same with FALSE
ex6$Eurozone[ex6$Eurozone==FALSE] <- "no"
```

7- In the column Country: how many country names from the list contain the letter "c" (capital- or lower-case) ?

Remember the grep function. Check the help page.

correction

```r
# country names with "c" (lower-case)
grep("c", ex6$Country)
# country names with "c" or "C" (ignoring case)
grep("c", ex6$Country, ignore.case = TRUE)
# show actual country names
grep("c", ex6$Country, value=TRUE, ignore.case = TRUE)
```

8- According to that data frame: how many people live: + in the European union (whole table) ? + in the Eurozone ?

correction

```r
# sum the whole population column
sum(ex6$Population)
# select elements of ex6 where Eurozone is "yes"
ex6$Eurozone == "yes"
# select only elements in Population for which the corresponding Eurozone elements are "yes"
ex6$Population[ex6$Eurozone == "yes"]
# sum that selection
sum(ex6$Population[ex6$Eurozone == "yes"])
```

9- Write ex6 into file ex6c_output.txt

After each of the following steps, check the output file in the RStudio file browser (lower-right panel).

- Try with the default arguments.

correction

```r
write.table(ex6, file="ex6c_output.txt")
```

- Add the argument "row.names" set to FALSE.

correction

```r
write.table(ex6, file="ex6c_output.txt",
            row.names = FALSE)
```

- Add the argument "quote" set to FALSE.

correction

```
write.table(ex6, file="ex6c_output.txt",
            row.names = FALSE,
            quote = FALSE)
```

- Add the argument "sep" set to "\t" or to","

correction

```
write.table(ex6, file="ex6c_output.txt",
            row.names = FALSE,
            quote = FALSE,
            sep="\t")
```

correction

```
write.table(ex6, file="ex6c_output.txt",
            row.names = FALSE,
            quote = FALSE,
            sep=",")
```

Chapter 12

Library and packages

- **Packages** are collections of R functions, data, and compiled code in a well-defined format.
- The directory where packages are stored is called the **library**.

Source of definitions: http://www.statmethods.net/interface/packages.html

12.1 R base

A set a standard packages which are supplied with R by default. Example: package base (write, table, rownames functions), package utils (read.table, str functions), package stats (var, na.omit, median functions).

12.2 R contrib

All other packages:

- CRAN: Comprehensive R Archive Network
  - 13735* packages available
  - find packages in https://cran.r-project.org/web/packages/
- Bioconductor:
  - 1649* packages available
  - find packages in https://bioconductor.org/packages

As of February 2019*

Bioconductor

Set of R packages specialized in the analysis of bioinformatics data.

Bioconductor supports most types of **genomics and NGS data** (e.g. limma, DESeq2, BayesPeak) and integrates: * Specific data classes (e.g. Granges from GenomicRanges) * Integrates command line tools (e.g Rsamtools) * Annotation tools (e.g. biomaRt)

There are different types of Bioconductor packages: * **Software**: set of functions + e.g. DESeq2 (NGS data analysis) * **Annotation**: annotation of specific arrays, organisms, events, etc. + e.g. BSgenome.Hsapiens.UCSC.hg38 * **Experiment**: data that can be loaded and used + e.g. ALL (acute lymphoblastic leukemia dataset)

12.3 Install a package

- With RStudio:
- From the console:

```
install.packages(pkgs="ggplot2")
```

- Install a bioconductor package:
  - For R version >= 3.5.0

```
# Install Bioconductor package manager
install.packages(pkgs="BiocManager")
# Install Bioconductor package
BiocManager::install("DESeq2")
```

```
+ For older R versions
```

```
# Source (load into environment) script containing biocLite function
source("http://www.bioconductor.org/biocLite.R")
# Use biocLite function to install Bioconductor package
biocLite("DESeq2")
```

12.4 Load a package

- With RStudio:
- From the console:

```
library("ggplot2")
```

12.5 Check what packages are currently loaded

```
sessionInfo()
```

```
## R version 3.3.2 (2016-10-31)
## Platform: x86_64-redhat-linux-gnu (64-bit)
## Running under: Scientific Linux 7.2 (Nitrogen)
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] ggplot2_2.2.1
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.2       rstudioapi_0.7   knitr_1.24       magrittr_1.5
##  [5] munsell_0.4.3    colorspace_1.3-2 rlang_0.4.0      stringr_1.4.0
##  [9] plyr_1.8.4       tools_3.3.2      grid_3.3.2       gtable_0.2.0
## [13] xfun_0.8         tinytex_0.15     htmltools_0.3.6  yaml_2.2.0
## [17] lazyeval_0.2.1   digest_0.6.20    tibble_2.1.3     crayon_1.3.4
## [21] bookdown_0.12    evaluate_0.14    rmarkdown_1.14.3 stringi_1.4.3
## [25] pillar_1.4.2     scales_0.5.0     pkgconfig_2.0.1
```

12.6 List functions from a package

- With RStudio

- From the console

```
ls("package:ggplot2")
```

12.7 RStudio server at CRG

If you can't install packages (permission issues), you first need to specify a writeable directory to install the packages into.

Follow the steps below:

```
# Go to your home directory
setwd("~")
# Create a directory where to store the packages
dir.create("R_packages")
# Add directory location to the library path
.libPaths("~/R_packages/")
```

12.8 Exercise 7: Library and packages

Create the script "exercise7.R" and save it to the "Rcourse/Module2" directory: you will save all the commands of exercise 7 in that script. Remember you can comment the code using #.

correction

```
getwd()
setwd("Rcourse/Module2")
setwd("~/Rcourse/Module2")
```

1- Install and load the packages ggplot2 and WriteXLS

correction

```
# Install the 2 packages at once
install.packages(pkgs=c("ggplot2", "WriteXLS"))
# Load in the environment (one by one)
library("ggplot2")
library("WriteXLS")
```

Check with sessionInfo() that the packages were loaded.

2- ggplot2 loads automatically the diamonds dataset in the working environment: you can use it as an object after ggplot2 is loaded.

What are the dimensions of diamonds? What are the column names of diamond?

correction

```
# Dimensions of diamonds
dim(diamonds)
# Column names of diamonds
colnames(diamonds)
```

You can read the help page of the diamonds dataset to understand what it contains!

Note: diamonds is a data frame: you can test it with is.data.frame(diamonds) (returns TRUE).

3- Select the columns carat, cut, color and price of diamonds and store in the object diams1.

correction

```
# Select columns
diams1 <- diamonds[,c("carat", "cut", "color", "price")]
```

4- Install and load the package dplyr from the Console.

correction

```
# Install package
install.packages(pkgs="dplyr")
# Load package
library("dplyr")
```

5- Use the function "sample_n" from the dplyr package to randomly sample 200 lines of diams1: save in diams object.

correction

```
# Subset data frame
diams <- sample_n(tbl=diams1, size=200)
```

-6.Save diams into 2 files:

- diamonds200.txt with write.table
- diamonds200.xls with WriteXLS

Note: read about and play with the different options of both functions and check the output files.

correction

```
# Write a text file with write.table
write.table(x=diams,
    file="diamonds200.txt",
        row.names=FALSE,
        quote=FALSE,
        sep="\t")
# Write an Excel file with WriteXLS
WriteXLS(x=diams,
    ExcelFileName="diamonds200.xls",
    row.names=FALSE,
    col.names=TRUE,
    FreezeRow=1,
    BoldHeaderRow=TRUE)
```

Chapter 13

Regular expressions

Regular expressions are tools to **describe patterns in strings**.

13.1 Find simple matches with grep

- Find a pattern anywhere in the string (outputs the index of the element):

```
# By default, outputs the index of the element matching the pattern
grep(pattern="Gen",
    x="Genomics")
```

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

- Show actual element where the pattern is found (instead of the index only) with **value=TRUE**:

```
# Set value=TRUE
grep(pattern="Gen",
        x="Genomics",
        value=TRUE)
```

```
## [1] "Genomics"
```

- Non case-sensitive search with **ignore.case=TRUE**:

```r
# Enter the pattern in lower-case, but case is ignored
grep(pattern="gen",
        x="Genomics",
        value=TRUE,
        ignore.case=TRUE)
```

```
## [1] "Genomics"
```

- Show if it DOESN'T match the pattern with **inv=TRUE**:

```r
# Shows what doesn't match
grep(pattern="gen",
        x="Genomics",
        value=TRUE,
        ignore.case=TRUE,
    inv=TRUE)
```

```
## character(0)
```

13.2 Regular expressions to find more flexible patterns

Special characters used for pattern recognition:

$ | Find pattern at the end of the string |
^ | Find pattern at the beginning of the string |
{n} | The previous pattern should be found exactly n times |
{n,m} | The previous pattern should be found between n and m times|
+ | The previous pattern should be found at least 1 time |
* | One or more allowed, but optional |
? | One allowed, but optional |

Match your own pattern inside []

$$abc$$

: matches a, b, or c. ^

$$abc$$

: matches a, b or c at the beginning of the element. ^A

$$abc$$

+: matches A as the first character of the element, then either a, b or c ^A

$$abc$$

*: matches A as the first character of the element, then optionally either a, b or c ^A

$$abc$$

{1}_: matches A as the first character of the element, then either a, b or c (one time!) followed by an underscore

$$a - z$$

: matches every character between a and z.

$$A - Z$$

: matches every character between A and Z.

$$0 - 9$$

: matches every number between 0 and 9.

- Match anything contained between brackets (here either g or t) at least once:

```
grep(pattern="[gt]+",
    x=c("genomics", "proteomics", "transcriptomics"),
    value=TRUE)
```

```
## [1] "genomics"        "proteomics"      "transcriptomics"
```

- Match anything contained between brackets at least once AND at the start of the element:

```
grep(pattern="^[gt]+",
        x=c("genomics", "proteomics", "transcriptomics"),
        value=TRUE)
```

```
## [1] "genomics"        "transcriptomics"
```

- **Create a vector of email addresses:**

```
vec_ad <- c("marie.curie@yahoo.es", "albert.einstein01@hotmail.com",
    "charles.darwin1809@gmail.com", "rosalind.franklin@aol.it")
```

- Keep only email addresses finishing with “es”:

```
grep(pattern="es$",
        x=vec_ad,
        value=TRUE)
```

```
## [1] "marie.curie@yahoo.es"
```

13.3 Substitute or remove matching patterns with gsub

From the same vector of email addresses:

- Remove the “@” symbol and the email provider from each address

```
gsub(pattern="@[a-z.]+",
        replacement="",
        x=vec_ad)
```

```
## [1] "marie.curie"        "albert.einstein01"  "charles.darwin1809"
## [4] "rosalind.franklin"
```

- Substitute the “@” symbol with “*at*”

```
gsub(pattern="@",
        replacement="_at_",
        x=vec_ad)
```

```
## [1] "marie.curie_at_yahoo.es"          "albert.einstein01_at_hotmail.com"
## [3] "charles.darwin1809_at_gmail.com"  "rosalind.franklin_at_aol.it"
```

- Substitute “es” and “it” by “eu”

```
gsub(pattern="es$|it$",
    replacement="eu",
    x=vec_ad)
```

```
## [1] "marie.curie@yahoo.eu"          "albert.einstein01@hotmail.com"
## [3] "charles.darwin1809@gmail.com"  "rosalind.franklin@aol.eu"
```

13.4 Predefined variables to use in regular expressions:

[:lower:] | Lower-case letters |
[:upper:] | Upper-case letters |
[:alpha:] | Alphabetic characters: [:lower:] and [:upper:] |
[:digit:] | Digits: 0 1 2 3 4 5 6 7 8 9 |
[:alnum:] | Alphanumeric characters: [:alpha:] and [:digit:] |
[:print:] | Printable characters: [:alnum:], [:punct:] and space. |
[:punct:] | Punctuation characters: ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [ ] ^ _ ` { | } ~ |
[:blank:] | Blank characters: space and tab |

- Take the previous character vector containing email addresses:
  - Remove the @ and the email provider from each address

```
gsub(pattern="@[[:lower:][:punct:]]+",
    replacement="",
    x=vec_ad)
```

```
## [1] "marie.curie"        "albert.einstein01"  "charles.darwin1809"
## [4] "rosalind.franklin"
```

```
* Same thing but remove additionally any number(s) BEFORE the @ (if any):
```

```
gsub(pattern="[[:digit:]]*@[[:lower:][:punct:]]+",
        replacement="",
        x=vec_ad)
```

```
## [1] "marie.curie"       "albert.einstein"   "charles.darwin"
## [4] "rosalind.franklin"
```

```
* Same but simplified:
```

```
gsub(pattern="[[:digit:]]*@[[:print:]]+",
        replacement="",
        x=vec_ad)
```

```
## [1] "marie.curie"       "albert.einstein"   "charles.darwin"
## [4] "rosalind.franklin"
```

13.5 Use grep and regular expressions to retrieve columns by their names

Example of a data frame:

```
# Build data frame
df_regex <- data.frame(expression1=1:4,
    expression2=2:5,
    expression3=4:7,
    annotation=LETTERS[1:4],
    expression4=6:3,
    average_expression=c(3.25, 3.75, 4.25, 4.75),
    stringsAsFactors=FALSE)

# Select column names that start with "expression"
grep(pattern="^expression",
    x=colnames(df_regex))
```

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

```
# Select columns from df_regex if their names start with "expression"
df_regex[, grep(pattern="^expression", colnames(df_regex))]
```

```
##   expression1 expression2 expression3 expression4
## 1           1           2           4           6
## 2           2           3           5           5
## 3           3           4           6           4
## 4           4           5           7           3
```

13.6 Exercise 8: Regular expressions

Create the script "exercise8.R" and save it to the "Rcourse/Module2" directory: you will save all the commands of exercise 8 in that script. Remember you can comment the code using #.

correction

```
getwd()
setwd("~/Rcourse/Module2")
```

1- Play with grep

- Create the following data frame

```
df2 <- data.frame(age=c(32, 45, 12, 67, 40, 27),
    citizenship=c("England", "India", "Spain", "Brasil", "Tunisia", "Poland"),
    row.names=paste(rep(c("Patient", "Doctor"), c(4, 2)), 1:6, sep=""),
    stringsAsFactors=FALSE)
```

Using grep: create a smaller data frame df3 that contains only the Patient but NOT the Doctor information.

correction

```
# Select row names
rownames(df2)
```

```
## [1] "Patient1" "Patient2" "Patient3" "Patient4" "Doctor5"  "Doctor6"
```

```
# Select only rownames that correspond to patients
grep("Patient", rownames(df2))
```

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

```
# Create data frame that contains only those rows
df3 <- df2[grep("Patient", rownames(df2)), ]
```

2- Play with gsub

Build this vector of file names:

```
vector1 <- c("L2_sample1_GTAGCG.fastq.gz", "L1_sample2_ATTGCC.fastq.gz",
    "L1_sample3_TGTTAC.fastq.gz", "L4_sample4_ATGGTA.fastq.gz")
```

Use gsub and an appropriate regular expression to remove all but "sample1", "sample2", "sample3" and "sample4" from vector1.

correction

```
# | is used as OR
gsub(pattern="L[124]{1}_|_[ATGC]{6}.fastq.gz",
    replacement="",
    x=vector1)
```

```
## [1] "sample1" "sample2" "sample3" "sample4"
```

Chapter 14

Repetitive execution

Loops are used to repeat a specific block of code.

Structure of the **for loop**:

```
for(i in vector_expression){
    action_command
}
```

3 main elements: * **i** is the loop variable: it is updated at each iteration. * **vector_expression**: value attributed to **i** at each iteration (the number of iterations is the **length of vector_expression**). * **action_command**: what is to be done at each iteration.

Note the usage of **curly brakets {}** to start and end the loop!

- Example:

```
for(i in 2:5){
    y <- i*2
    print(y)
}
```

```
## [1] 4
## [1] 6
## [1] 8
## [1] 10
```

- Example of a **for loop** that iterates over a character vector:

```
# Character vector
myfruits <- c("apple", "pear", "grape")
# For loop that prints the current element and its number of characters
for(j in myfruits){
    print(j)
    print(nchar(j))
}
```

```
## [1] "apple"
## [1] 5
## [1] "pear"
## [1] 4
## [1] "grape"
## [1] 5
```

- Example of a **for loop** that iterates over each row of a matrix, and prints the minimum value of that row :

```
# Matrix
mymat <- matrix(rnorm(800),
    nrow=50)
# For loop over mymat rows
for(i in 1:nrow(mymat)){
    print(i)
    print(min(mymat[i,]))
}
```

```
## [1] 1
## [1] -2.281851
## [1] 2
## [1] -1.278704
## [1] 3
## [1] -1.758846
## [1] 4
## [1] -1.086473
## [1] 5
## [1] -2.261945
## [1] 6
## [1] -2.105386
## [1] 7
## [1] -1.631014
## [1] 8
## [1] -1.735083
## [1] 9
```

```
## [1] -1.849288
## [1] 10
## [1] -2.04397
## [1] 11
## [1] -1.892094
## [1] 12
## [1] -1.607064
## [1] 13
## [1] -1.997987
## [1] 14
## [1] -1.898659
## [1] 15
## [1] -2.669472
## [1] 16
## [1] -1.749975
## [1] 17
## [1] -0.9498798
## [1] 18
## [1] -1.124963
## [1] 19
## [1] -0.6238186
## [1] 20
## [1] -1.569462
## [1] 21
## [1] -0.7128029
## [1] 22
## [1] -1.613293
## [1] 23
## [1] -2.531049
## [1] 24
## [1] -1.633086
## [1] 25
## [1] -1.026743
## [1] 26
## [1] -1.191333
## [1] 27
## [1] -1.650943
## [1] 28
## [1] -2.006542
## [1] 29
## [1] -1.542367
## [1] 30
## [1] -1.556476
## [1] 31
## [1] -1.819371
## [1] 32
```

```
## [1] -1.141169
## [1] 33
## [1] -1.515073
## [1] 34
## [1] -2.740146
## [1] 35
## [1] -1.687516
## [1] 36
## [1] -1.543791
## [1] 37
## [1] -1.069926
## [1] 38
## [1] -2.173883
## [1] 39
## [1] -1.725881
## [1] 40
## [1] -1.538689
## [1] 41
## [1] -0.3736716
## [1] 42
## [1] -1.685601
## [1] 43
## [1] -1.973906
## [1] 44
## [1] -2.458968
## [1] 45
## [1] -2.028095
## [1] 46
## [1] -2.060121
## [1] 47
## [1] -1.51852
## [1] 48
## [1] -1.353344
## [1] 49
## [1] -1.310166
## [1] 50
## [1] -1.880703
```

14.1 Exercise 9: For loop

Create the script "exercise9.R" and save it to the "Rcourse/Module2" directory: you will save all the commands of exercise 9 in that script. Remember you can comment the code using #.

correction

```
getwd()
setwd("~/Rcourse/Module2")
```

1- Write a for loop that iterates over 2 to 10 and prints the square root of each number (function sqrt()).

correction

```
for(i in 2:10){
    print(sqrt(i))
}
```

```
## [1] 1.414214
## [1] 1.732051
## [1] 2
## [1] 2.236068
## [1] 2.44949
## [1] 2.645751
## [1] 2.828427
## [1] 3
## [1] 3.162278
```

2- Write a for loop that iterates over 5 to 15 and prints a vector of 2 elements containing each number and its square root

correction

```
for(i in 5:15){
    veci <- c(i, sqrt(i))
        print(veci)
}
```

```
## [1] 5.000000 2.236068
## [1] 6.00000 2.44949
## [1] 7.000000 2.645751
## [1] 8.000000 2.828427
## [1] 9 3
## [1] 10.000000  3.162278
## [1] 11.000000  3.316625
## [1] 12.000000  3.464102
## [1] 13.000000  3.605551
## [1] 14.000000  3.741657
## [1] 15.000000  3.872983
```

3- Create the following matrix

```
mat1 <- matrix(rnorm(40), nrow=20)
```

- Write a for loop that iterates over each row of mat1 and prints the median value of each row.

correction

```
for(j in 1:nrow(mat1)){
    # extract the row
    rowj <- mat1[j,]
    # print rowj
    print(rowj)
    # print median value in row
    print(median(rowj))
}
```

```
## [1] -1.426074  1.567936
## [1] 0.07093107
## [1] -0.2923305 -0.7264357
## [1] -0.5093831
## [1]  0.2802764 -0.5765799
## [1] -0.1481518
## [1] -0.8333338  0.1189056
## [1] -0.3572141
## [1] -0.2053112 -1.1162466
## [1] -0.6607789
## [1] -0.9001538  0.6344393
## [1] -0.1328572
## [1] -0.1561286  0.7755513
## [1] 0.3097114
## [1]  1.1316134 -0.4716209
## [1] 0.3299963
## [1] 0.3560698 1.5721636
## [1] 0.9641167
## [1] 0.2666173 0.2086493
## [1] 0.2376333
## [1]  0.63159641 -0.08313833
## [1] 0.274229
## [1] -0.1191863 -0.2183160
## [1] -0.1687511
## [1] -0.07855972  0.17627241
## [1] 0.04885635
## [1] -0.7207866 -0.3078621
## [1] -0.5143243
```

```
## [1] -0.7787821 -1.1064297
## [1] -0.9426059
## [1] -1.602712  1.150602
## [1] -0.2260552
## [1]  0.09832890 -0.03826973
## [1] 0.03002959
## [1] 0.2754761 1.5711289
## [1] 0.9233025
## [1]  1.16296099 -0.04964327
## [1] 0.5566589
## [1]  0.1836257 -1.0311462
## [1] -0.4237603
```

Chapter 15

Conditional statement

"if" statement

Structure of the **if statement**:

```
if(condition){
    action_command
}
```

If the **condition** is TRUE, then proceed to the **action_command**; if it is FALSE, nothing happens.

```
k <- 10
# print if value is > 3
if(k > 3){
  print(k)
}
```

```
## [1] 10
```

```
# print if value is < 3
if(k < 3){
  print(k)
}
```

With **else**

```
if(condition){
    action_command1
```

```
}else{
    action_command2
}
```

If the **condition** is TRUE, then proceed to the **action_command1**; if the **condition** is FALSE, proceed to **action_command2**.

```
k <- 3
if(k > 3){
  print("greater than 3")
}else{
  print("less than 3")
}
```

```
## [1] "less than 3"
```

With **else if**

```
if(condition1){
    action_command1
}else if(condition2){
    action_command2
}else{
    action_command3
}
```

If the **condition1** is TRUE, then proceed to the **action_command1**; if the **condition1** is FALSE, test for **condition2**: if the **condition2** is TRUE, proceed to the **action_command2**; if neither **condition1** nor **condition2** are TRUE, then proceed to the **action_command3**. *Note that you can add up as many **else if** statements as you want.*

- Example without **else**

```
k <- -2
# Test whether k is positive or negative or equal to 0
if(k < 0){
    print("negative")
}else if(k > 0){
    print("positive")
}else if(k == 0){
    print("is 0")
}
```

```
## [1] "negative"
```

- Example with **else**

```
k <- 10

# print if value is <= 3
if(k <= 3){
  print("less than or equal to 3")
}else if(k >= 8){
  print("greater than or equal to 8")
}else{
  print("greater than 3 and less than 8")
}
```

```
## [1] "greater than or equal to 8"
```

- **If statement** in **For loop**:

```
# Matrix
mymat <- matrix(rnorm(800),
        nrow=50)

# Loop over rows of mymat and print row if its median value is > 0
for(i in 1:nrow(mymat)){
    # extract the current row
    rowi <- mymat[i,]
    # if median of row is > 0, print row
    if(median(rowi) > 0){
        print(rowi)
    }
}
```

```
##  [1]  0.9163559  0.7349928 -0.3772329  1.8846106  1.8660998 -0.4275973
##  [7] -0.9926951 -0.6496524 -1.3389813 -0.2507018  1.0190404  1.2624915
## [13]  0.2633816  0.5655954  0.9427435 -1.3802493
##  [1]  0.1479990  0.3879150 -0.7901787 -0.3531613 -0.6177629  1.6146533
##  [7]  1.4207582  0.0679865 -1.9111646  2.3818324  0.8704155  0.2746928
## [13]  0.4210446 -0.2882419  2.0833951  1.1803731
##  [1] -1.18213195  1.07708663 -0.92247614  0.10125880  3.30162107
##  [6]  0.03840533  0.20534091 -1.02017511 -0.19410326  1.49001195
## [11]  0.13081432 -0.98758274  0.07946315 -0.17199232 -0.19939657
## [16]  0.16472811
##  [1]  1.4903296  0.8809405  0.1759845 -0.5655213  0.1457286  0.7656389
```

```
##  [7]  0.5090503  0.3663223 -1.8027452 -0.1507720  0.7427838 -0.1407290
## [13]  1.2303268  0.4318421 -0.9135202 -0.6510606
##  [1]  0.06466694  1.63324127 -0.12953666 -0.26742867  1.00173669
##  [6]  0.65061870  0.41818662 -0.14009611  0.49238252  1.05598027
## [11] -0.55235687 -1.94123942  0.86880122  0.10000467 -0.43941882
## [16] -0.75603765
##  [1] -1.7422557 -0.5463444 -2.1947102  1.2016771  0.3473332  0.7490919
##  [7]  0.9560989  0.2476182  0.0942439  0.7720824  0.3834482 -1.0109472
## [13] -0.2040869  0.2547889 -1.0169789  0.0888180
##  [1]  1.06876233  0.81086517  0.06373738  1.66755943 -0.49338925
##  [6] -0.07320543  1.38823193  1.45227863  0.59236036 -0.58559162
## [11] -0.27949354 -0.30149370  2.36484144 -0.54134386 -0.86756969
## [16]  1.33858747
##  [1] -0.00187001 -0.55568538 -0.99610080  1.72784659  0.05502652
##  [6] -0.33081174  0.25796894  1.00000955  0.11414184  1.63564941
## [11]  0.70643683 -0.56096678  1.14167762 -0.09289810 -0.06743940
## [16] -0.67861355
##  [1] -0.67715967 -1.69349575  0.30382420  1.52126875  0.52217950
##  [6]  0.46763744 -0.11964099  0.35041537  0.53041949  0.50046232
## [11]  1.33281160 -1.11929100  2.04234456  0.81692747 -0.96672225
## [16]  0.04917855
##  [1]  1.298443420  0.827175177  0.668623641 -0.007582964  0.137749387
##  [6] -1.208501386 -0.230904369  0.324687007  0.064489634  0.659187294
## [11] -0.054669213 -0.336844682 -1.127411201 -0.537984204  1.260017569
## [16]  0.188788949
##  [1] -1.6534706  0.7547948  1.5782940  0.2718084 -1.1491107 -0.0855659
##  [7] -0.9590750  1.4911942  1.8661906 -0.5464456 -1.7640418 -1.0713838
## [13]  0.2298441  1.0284024  0.9106712  0.6632802
##  [1]  0.78252442  0.65076205 -1.52821200  0.01493103  1.19286734
##  [6]  1.76112571 -0.30045810 -0.63845535 -1.73530526  1.60211864
## [11]  1.65519109 -0.56811726 -0.29404688  1.69879580  0.71160065
## [16]  1.58981886
##  [1]  1.03244700  0.41652054 -0.11148977  0.07397958  0.65159507
##  [6]  0.33062436  0.08272582 -1.65712107 -3.28440112  0.48075125
## [11] -1.07788982  0.83927262  1.42513216 -3.09908994  0.55946365
## [16]  0.68637454
##  [1] -0.35627905 -0.61998336 -1.66788721 -0.50768166  1.01260140
##  [6]  1.46236319  0.01026259  1.50641234  0.52185060 -0.51596638
## [11]  1.92549261  0.66663360  1.40124385  0.15986504 -1.03286094
## [16] -0.72861591
##  [1]  0.3267208 -0.8572655 -0.2227745  0.3381801 -1.8095795  0.9173742
##  [7]  2.4453909  0.9631062 -0.7604507  1.1087977 -0.8251043  0.9018535
## [13]  0.7032885  0.9434812  1.2558652  0.3437966
##  [1] -1.6063631  0.6190595  0.4328564 -0.9252788  2.0502140 -0.5830851
##  [7]  1.0150514 -1.1490475 -0.6375838 -2.3441523  0.5580132  1.7991886
## [13]  0.1259076  1.2699128  1.1202740 -0.9330312
```

```
##  [1]  1.0610155  1.0304131 -0.4212878  0.8855348  0.9313333  0.9831432
##  [7]  0.3791134 -0.5005799  1.2897054 -0.8858256  2.2115244 -0.5727546
## [13]  1.2246651  0.1696573  0.5670294 -0.1406598
##  [1] -0.5718241  0.2876528 -0.7599872 -1.3426841 -1.4715546  0.2878218
##  [7]  1.3433642 -0.1256095  1.3973283  1.0172665  0.6500409 -0.7348384
## [13]  0.6617577 -0.7528288  1.0778910  0.9264117
##  [1] -0.31262493 -0.99195330  0.78965061 -0.87232147 -1.42011913
##  [6]  0.24568271  1.55472827 -0.91171218 -0.02790880 -0.04541801
## [11] -0.74139419  0.99752304  0.97987161  0.95292547  0.57179762
## [16]  0.19768017
##  [1] -1.6842057  1.2720440  0.4180212  1.5457446  0.1939920 -1.5472676
##  [7]  1.1827240  0.8265619 -0.2319350 -1.3576666  2.6136101 -0.2607527
## [13] -2.3777760  0.3662957  0.4050278 -0.4598094
##  [1] -0.85276625 -0.35827999  0.53089747  1.11690665  1.17642620
##  [6]  0.30390853 -1.32593018  0.47064455  0.08246726 -0.35423450
## [11] -0.13982965  0.80719698  0.39038268  2.11009152 -1.75288717
## [16] -0.10843135
##  [1] -1.13235048  1.49940993 -0.45705973  1.17642667  0.85525618
##  [6] -0.75936937  0.13586177  0.55534466  0.05689966  0.78617742
## [11]  0.76277994  2.13964263 -2.16606066 -0.63659314 -0.87816197
## [16]  0.06439538
##  [1]  1.3351701  2.4924504  0.1728600  0.8903435  0.8725524  0.5655810
##  [7]  0.9924199 -0.5826830  0.8207116  0.1220051 -0.2526791 -1.0481275
## [13] -1.6940530 -0.3888586 -0.4922859  1.6256465
```

15.1 Exercise 10: If statement

Create the script "exercise10.R" and save it to the "Rcourse/Module2" directory: you will save all the commands of exercise 10 in that script. Remember you can comment the code using #.

correction

```
getwd()
setwd("~/Rcourse/Module2")
```

1- Create vector vec2 as:

```
vec2 <- c("kiwi", "apple", "pear", "grape")
```

- Use an if statement and the %in% function to check whether "apple" is present in vec2 (in such case print "there is an apple!")

correction

```
if("apple" %in% vec2){
    print("there is an apple there")
}
```

```
## [1] "there is an apple there"
```

- Use an if statement and the %in% function to check whether “grapefruit” is present in vec2: if “grapefruit” is not found, test for a second condition (using **else if**) that checks if “pear” is found.

correction

```
if("grapefruit" %in% vec2){
        print("there is a grapefruit there")
}else if("pear" %in% vec2){
    print("there is no grapefruit but there is a pear")
}
```

```
## [1] "there is no grapefruit but there is a pear"
```

- Add an **else** section in case neither grapefruit nor pear is found in vec2. Test your **if** statement also on vec3:

```
vec3 <- c("cherry", "strawberry", "blueberry", "peach")
```

correction

```
if("grapefruit" %in% vec2){
        print("there is a grapefruit there")
}else if("pear" %in% vec2){
        print("there is no grapefruit but there is a pear")
}else{
    print("no grapefruit and no pear")
}
```

```
## [1] "there is no grapefruit but there is a pear"
```

2- If statement in for loop

Create the following matrix:

```
mat4 <- matrix(c(2, 34, 1, NA, 89, 7, 12, NA, 0, 38),
    nrow=5)
```

Loop over rows with **for** of mat4 and print row number and entire row **if** you find an NA.

correction

```
for(k in 1:nrow(mat4)){
    # extract row
    rowk <- mat4[k,]
    if(any(is.na(rowk))){
        print(k)
        print(rowk)
    }
}
```

```
## [1] 3
## [1]  1 NA
## [1] 4
## [1] NA  0
```

3- For loop, if statement and regular expression

Create vector vec4 as:

```
vec4 <- c("Oct4", "DEPP", "RSU1", "Hk2", "ZNF37A", "C1QL1", "Shh", "Cdkn2a")
```

Loop over each element of "vec4": * If the element is a **human gene (all upper-case characters)**, print a vector of two elements: the name of the gene and "human gene". * If the element is a **mouse gene (only the first character is in upper-case)**, print a vector of two elements: the name of the gene and "mouse gene".

> Tip 1: *Use grep and a regular expression in the if statement !* Tip 2: *When grep does not find a match, it returns an element of **length 0** !* Tip 3: *You can also use grepl: check the help page*

correction

```
for(gene in vec4){
    if(length(grep(pattern="^[A-Z0-9]+$", x=gene)) != 0){
        print(c(gene, "human gene"))
    }else if(length(grep(pattern="^[A-Z]{1}[a-z0-9]+$", x=gene)) != 0){
```

```
        print(c(gene, "mouse gene"))
    }
}
```

```
## [1] "Oct4"       "mouse gene"
## [1] "DEPP"       "human gene"
## [1] "RSU1"       "human gene"
## [1] "Hk2"        "mouse gene"
## [1] "ZNF37A"     "human gene"
## [1] "C1QL1"      "human gene"
## [1] "Shh"        "mouse gene"
## [1] "Cdkn2a"     "mouse gene"
```

```
# With grepl
for(gene in vec4){
        if(grepl(pattern="^[A-Z0-9]+$", x=gene)){
                print(c(gene, "human gene"))
        }else if(grepl(pattern="^[A-Z]{1}[a-z0-9]+$", x=gene)){
                print(c(gene, "mouse gene"))
        }
}
```

```
## [1] "Oct4"       "mouse gene"
## [1] "DEPP"       "human gene"
## [1] "RSU1"       "human gene"
## [1] "Hk2"        "mouse gene"
## [1] "ZNF37A"     "human gene"
## [1] "C1QL1"      "human gene"
## [1] "Shh"        "mouse gene"
## [1] "Cdkn2a"     "mouse gene"
```