Exercise 1 (Install R and R-studio)

R is a open-source software (with GPL licence) and can be accessed freely for all platforms (Windows, Mac OS, Linux). You can download it under

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

A "professional" way of working with R is to edit R-scripts in an editor and to transfer the written code to a running R process. There are lots of editors supporting this. We recommend to use R-studio which is available under

http://rstudio.org.

(a) In order to install R, go to:

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

and click on the download button for your platform.
For the `Windows users`, click on "install R for the first time" and then "Download R 3.5.1 for Windows". Run it and save R in the default directory.
For the `Mac OS users`, click on "R-3.5.1.pkg" and download the R installer package. Run it and save R in the default directory.

(b) **After** you have installed R, you can install R-studio:

https://www.rstudio.com/products/rstudio/download/.

Down at the bottom there are "installers for supported platforms". Choose version 1.1.456. Follow the installation instructions. You can use the default settings in the setup-assistant.

Exercise 2 (Become familiar with R and R-Studio)

After you have successfully installed R and R-Studio, double click on the R-Studio icon. We will use R within the R studio envionment. We don't have to load R, R-studio loads it automatically.
You should now see three panes. On the left, there is the console (R with a command line). On the upper right you see the workspace (which is empty in the beginning). On the bottom right, there is a pane with five different windows: The file explorer (Files), a graphics window (Plots), an overview of installed R packages (Packages), a helper documentation (Help) and a viewer (Viewer).
We will now go through some R basics:

(a) In a very first step, type `3+3` in the R-console and press <Enter> to obtain the result. Then, try to solve the following tasks in R:

- $4 * 2 + 1$, $(2 * 4)/3$

```
4*2+1

## [1] 9

(2*4)/3

## [1] 2.666667
```

- 4^2, 10^2

```
4^2

## [1] 16

10^2

## [1] 100
```

- $log(1)$, e^1, $log(e^1)$

```
log(1)

## [1] 0

exp(1)

## [1] 2.718282

log(exp(1))

## [1] 1
```

(b) R is vector-oriented which is why we have to become familiar with the concept of a vector in R. In order to create a vector, we use the assignment operator <- or = (shortcut: press <Alt> and - simultaneously).

- Type `x <- 2` into your console and press <Enter>. The assignment operator has created an R object `x` that should now be visible in your R-environment (top right). `x` is a vector with one element.
  Create a second vector `y <- 3`. You can consider the elements of a vector by simply typing the name of the vector and <Enter>.

```
# Create the objects x and y
x <- 2
y <- 3
```

- Now, we can use the vectors for simple calculations as summarized below:
  - $x + y$, x/y, $x - y$

```
x+y
## [1] 5
x/y
## [1] 0.6666667
x-y
## [1] -1
```

- $log(x)$, x^2

```
log(x)
## [1] 0.6931472
x^2
## [1] 4
```

- Create the following objects:

```
x <- c(5,2,1,4,3)
y <- rep(1,5)
names <- c("Hans","Axel","Steph")
```

```
x <- c(5,2,1,4,3)
y <- rep(1,5)
names <- c("Hans","Axel","Steph")
```

Consider the objects and find out what the following commands are doing. If you want to know how, e.g. the `rep()` function, works, you can type `help(rep)` or `?rep`:

- `class(x); class(names); str(y)`

```
# Class gives the class of the object: numeric and character
class(x)
## [1] "numeric"
class(names)
## [1] "character"
# Str displays the structure of an object. It additionally contains the class
str(y)
##  num [1:5] 1 1 1 1 1
```

- `sum(x); mean(x); length(y); max(x)`

```
# Here we calculate the

# sum over all elements in x
sum(x)
```

```
## [1] 15
```

```
# the mean over all elements in x
mean(x)
```

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

```
# the number of elements in the vector y
length(y)
```

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

```
# The highest value in x
max(x)
```

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

- `names=="Axel"; names[3]; names[3]="Stef"`

```
# The == tests if the elements are equal to "Axel"
# and returns the values TRUE/FALSE
names=="Axel"
```

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

```
# The quared brackets allow to get the third element
# of the vector
names[3]
```

```
## [1] "Steph"
```

```
# With the = operator we assign a new element to the vector.
# It is equivalent to <-.
# Here we even overwrite the third element of the vector.
names[3]="Stef"
names
```

```
## [1] "Hans" "Axel" "Stef"
```

- $x[2];\ x[2] * y[1];\ x[2:4] + 1$

```
# We extract the second element of the vector
x[2]
```

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

```
# multiply the second element of x with the
# first element of y
x[2]*y[1]
```

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

```
# Add a +1 to the second, third and fourth element
# of the vector
x[2:4]+1
```

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

- $x <= 2;\ x[x <= 2];\ y! = 1$

```
# Test which elements of x als smaller or equal to 2
x<=2
```

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

```
# Return the elements of x that are smaller or
# equal to 2
x[x<=2]
```

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

```
# Test if the elements of y are not equal to 1
y!=1
```

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

Exercise 3 (Working with R scripts)

So far, we typed the commands directly in the R-console. If we want to save our code we can use R-scripts.

(a) Create a new script file via `File -> New File -> R Script`. You should now see four panes. Save the file as tutorial.R via `File -> Save` in your prefered directory. From now on, all your R instructions should be typed in this script-file. Make sure to comment your code (with the symbol #) as you go on.

(b) In the editor pane tutorial.R, create a new vector `z <- c(8,13,21)` as first line and `2*z` as second line:

```
z <- c(8,13,21)
2*z
```

```
## [1] 16 26 42
```

You have now several options to send your R code to the console:

- click on *Source* (upper right corner of the editor pane). All your code is sent to the command line but you will get no output.
- Point to the first line and click on *Run* (in the upper right corner of the editor pane). Only the first line is sent to the command line and evaluated. The cursor now points on the next line (second). Redo to send the second line to the R-console, and so on.
- If you want to run the first and the second line, select both lines and click *Run*.
- Instead of clicking on *Run* you can use the shortcut <Ctrl> + <Enter> (you have to press both keys at the same time). Both are equivalent.

Exercise 4 (Dataframe)

The goal of this exercise is to become acquainted with dataframes in R and to start to analyze data. The dataset which is used for this exercise can be found under

https://polybox.ethz.ch/index.php/s/bA45tIopHB7dBRX.

It is comprised of measurements from a survey of school children and stored in CSV format.

(a) R can read many different file formats either through base functionality or via the many packages available on CRAN. In case of CSV files, the `read.table()` function can be used. You can either read the data directly from an URL or download it and read it from the local file-system. The direct access link to the file for this exercise is

`https://polybox.ethz.ch/index.php/s/bA45tI0pHB7dBRX/download`.

Read the data into R. You can use: `dat <- read.table(..., sep=";", header=TRUE)`. The `sep` operator tells R that the CSV file is separated by ";". The `header=TRUE`, tells R that the first line of the CSV-file contains the header.

```
# I defined dir <- 'C:/.../' earlier. This has to be
# the COMPLETE path to the your file.
# setwd() changes your current working directory, i.e.
# the folder where R is working on and where it expects the data.
setwd(dir)
dat <- read.table(file = paste0(dir,"data/Erhebung_WI12t.csv"),
                  sep = ";", header = TRUE)
# paste0() combines dir and data/Erhebung_WI12t.csv to
# 'C:/.../data/Erhebung_WI12t.csv', i.e. it is the same as
# dat <- read.table(file = "C:/.../data/Erhebung_WI12t.csv",
```

```
#                   sep = ";", header = TRUE)

# You can also use the URL. BUT, the link has to be such that the
# data is directly downloaded (i.e. if you click on the link
# the data has to be downloaded directly. Otherwise it doesn't work!)
# url <- "https://polybox.ethz.ch/index.php/s/bA45tI0pHB7dBRX/download"
# dat <- read.table(file = url,
#                   sep = ";", header = TRUE)
```

(b) Inspect the object representing the dataset that you imported before. In particular, you should be able to answer the following questions (Hints: `names()`, `head()`, `str()`, `class()`, `dim()`)

- What are the names of the variables?

```
names(dat)
```

```
##  [1] "Age"                              "Sex"
##  [3] "Eye.color"                        "Hair.color"
##  [5] "Height"                           "Arm.span"
##  [7] "Hand.span"                        "Siblings"
##  [9] "Sports.team"                      "Handedness"
## [11] "Previous.statistical.knowledge"
```

- By which data type are the variables represented?

```
str(dat)
```

```
## 'data.frame': 18 obs. of  11 variables:
##  $ Age                           : int  26 24 25 26 25 23 23 24 22 23 ...
##  $ Sex                           : Factor w/ 2 levels "f","m": 2 2 2 2 2 2 2 2 2 2
##  $ Eye.color                     : Factor w/ 5 levels "blue","blue/gray",..: 1 3
##  $ Hair.color                    : Factor w/ 2 levels "blond","brown": 2 2 2 1 2
##  $ Height                        : int  173 183 183 174 178 189 185 178 177 186
##  $ Arm.span                      : int  174 181 178 178 180 183 184 177 174 183
##  $ Hand.span                     : num  21 23 23 23 21 22.5 24.5 22.5 23 24 ...
##  $ Siblings                      : Factor w/ 5 levels "?","1","2","3",..: 2 5 2 1
##  $ Sports.team                   : Factor w/ 2 levels "no","yes": 2 2 1 2 2 2 2 1
##  $ Handedness                    : Factor w/ 2 levels "left","right": 2 2 2 2 1 1
##  $ Previous.statistical.knowledge: Factor w/ 2 levels "little","substantial": 1 1

# We have integers, factors and numerical variables
```

- To which class belongs the created R-object?

```
class(dat)
```

```
## [1] "data.frame"
```

- How many rows and columns has the dataset?

```
# 18 rows, 11 columns
dim(dat)
```

```
## [1] 18 11
```

- How can the column `Arm.span` be accessed?
(Hint: dat$variablename or `dat[,"variable-name>"]`)

```
dat$Arm.span
```

```
##  [1] 174 181 178 178 180 183 184 177 174 183 190 155 180 184 181 197 165
## [18] 175
```

```
#or
dat[,'Arm.span']
```

```
##  [1] 174 181 178 178 180 183 184 177 174 183 190 155 180 184 181 197 165
## [18] 175
```

```
# or:
dat[,6]
```

```
##  [1] 174 181 178 178 180 183 184 177 174 183 190 155 180 184 181 197 165
## [18] 175
```

```
# The last one is not recommended, it's always better to
# access columns by name in case the order of the columns
# changes
```

- Print the third data-point.

```
dat[3,]
```

```
##   Age Sex   Eye.color Hair.color Height Arm.span Hand.span Siblings
## 3  25   m green/brown      brown    183      178        23        1
##   Sports.team Handedness Previous.statistical.knowledge
## 3          no      right                         little
```

- How old is the oldest student?

```
(max_age <- max(dat$Age))
```

```
## [1] 29
```

- How many students are 29 years old? (Hint: `table()`, `sum()`)

```
table(dat$Age)
```

```
##
## 21 22 23 24 25 26 27 28 29
##  1  1  4  2  4  2  1  1  2
```

```
sum(dat$Age == 29)
```

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

(c) Visualize the data `Arm.span` in a histogram. Try different bandwidths (Hint: `hist(..., breaks=)`). Don't forget to label the axes.

```
par(mfrow=c(1,3))
hist(dat$Arm.span, breaks = 5, xlab = "Arm span",
     main = "Histogram of Arm span (breaks=5)")
hist(dat$Arm.span, breaks = 10, xlab = "Arm span",
     main = "Histogram of Arm span (breaks=10)")
hist(dat$Arm.span, breaks = 20, xlab = "Arm span",
     main = "Histogram of Arm span (breaks=20)")
```