Seminar 1

1. Try using R as a calculator. Calculate these formulas and take note of the results.

```
1 + 1
1 * 0
1 / 0  # R knows limits

2 ^ 10
2**10

sqrt(10)
10 ^ (1 / 2)

pi
sqrt((42 + 4.2) ^ 2 + sin(exp(1) * pi))

(3 + 2i) / (2i) # R can work with imaginary numbers

i
1i
1*i
# i can 't stand alone, always has to be accompanied
    by a number
```

2. Display the documentation for a few R functions. Note how is the help document organised.

```
?sin
?load
?lm
?mean
example(mean)
example(lm) # might be useful in following lectures
```

3. Assign names to variables.

```
a <- 5
5 -> a
a = 5 # proper syntax uses <- instead of =

b <- 8
c <- a + b
c
```

```
(c <- a + b)
```

4. Remember that R is case senstitive.

```
a <- 25
A <- 4
a
A
# R is case sensitive

A == a
a <- A
a
A
A == a
```

5. Experiment with the usage of R logical operators.

```
1 == 1 # equal
1 == 2
1 != 1 # not equal
1 != 2
1 < 2
1 <= 2
TRUE & TRUE # and
TRUE | FALSE # or
(1 <= 2) & (2 >= 0) # combining
!FALSE # negating
!(2 < 1)
```

6. Get familiar with different data types in R.

```
# numerical - use decimal point
a <- 2.2

# string - both single and double quotation marks
   can be used,
# as long as you use the same at both ends
a <- "something"
b <- 'something else'

# logical - TRUE / FALSE or T/F can be used
a <- TRUE
b <- T
```

```
c <- FALSE
d <- F

# can be added up - TRUE is 1, FALSE is 0
a + b

is.integer(a)
is.integer(a + b) # we changed the data type
```

7. Get familiar with the most simple data structure in R - **vectors**.

   (a) Create a vector of numbers 1 to 4.

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

   (b) Create a vector of which repeats numbers.

   ```
   ?rep

   rep(1, 10)

   rep(1:3, 10)
   rep(1:3, times = 10)
   rep(1:3, each = 10)
   ```

   (c) Create sequences of numbers. Try creating sequences of specific lengths or with specific differences between consecutive values.

   ```
   ?seq

   seq(from = 1, to = 4)
   seq(1, 4)
   seq(to = 4, from = 1)

   seq(from = 1, to = 10, length = 5)
   seq(1, 10, length = 5)
   seq(1, 10, by = 2)
   seq(from = 1, by = 2, length = 5)
   ```

   (d) Create a vector of random numbers.

   ```
   runif(10) # uniformly distributed on [0; 1]
   runif(10, min = 0, max = 10) # uniformly
      distributed between 0 and 10
   ```

(e) Create a non-numerical vector.

```
(character.vector <- c('this', 'is', 'a', '
   vector', 'too'))
(logical.vector <- c(TRUE, FALSE, FALSE))

(mixed.vector1 <- c(1, TRUE, 5))
# in this case the resulting vector is taken to
   be numerical (TRUE = 1)

(mixed.vector2 <- c(1, TRUE, 'a'))
# in this case the resulting vector is taken to
   be string

(mixed.vector3 <- c(1, 2, 1.5))
```

(f) Create a new data sample based on your vector.

```
data.vector <- c("low", "medium", "high")

sample(data.vector, replace = TRUE, size = 15)
# chooses 15 values from the data.vecotor with
   replacement
# the option to replace values must be set to
   TRUE if we are creating
# a sample larger than our original data vector

sample(data.vector, replace = TRUE, size = 15,
       prob = c(0.1, 0.2, 0.7)) # add
          probabilities to sampling
sample(data.vector, replace = FALSE, size = 2)
```

(g) Pull elements out of a vector.

```
t <- 1:30

t[1]          # first element, R indexes from 1
t[1:3]        # first 3 elements
t[c(1, 7)]  # 1st and 7th element
t[seq(from = 1, by = 2, to = length(t))] #
   elements on odd positions

head(t, 10)# first 10 elements
tail(t, 3) # last 3 elements
```

(h) Subset a vector using logical operators.

```
t[rep(c(TRUE, FALSE), each = 15)]

1 > 10
2 > 10
c(1, 2) > 10          # R can compare vectors
c(1, 2) > c(3, 1)     # element by element

t[t > 10]
t[t > 10 & t < 25]
t[t > 10 | t < 25]
```

8. Get to know another data structure in R - **factors**. Factors are used to store categorical variables. The distinct categories are called *levels*.

   (a)
   ```
   f <- factor(1:3)
   1 + 2
   f[1] + f[2]
   # factor understands its values as distinct
      categories,
   # they no longer abide the rules of a string or
      numeric variable

   x <- sample(c('child', 'teenager', 'adult'),
      replace = TRUE, size = 15)
   (f <- factor(x)) # default order of levels is
      alphabetical

   f <- factor(f, levels = c('child', 'teenager', '
      adult')) # reordering a factor
   levels(f) # can be useful in following seminars
   ```

9. Get familiar with the another data structure in R - **matrices**.

   (a) Create a matrix, name the rows and columns.

   ```
   matrix(c(1:20), ncol = 4) # specify number of
      rows
   matrix(1:5, ncol = 3, nrow = 5) # specify number
       of rows and columns
   matrix(c(1:20), ncol = 4, byrow = T)
   rbind(1:3, c(4, 8, 9), c(11, 21, 23)) # bind
      rows
   cbind(1:3, c(4, 8, 9), c(11, 21, 23)) # bind
      columns

   # adding column or row names
   G <- cbind(1:3, c(4, 8, 9), c(11, 21, 23))
   colnames(G) <- c('col1', 'col2', 'col3')
   rownames(G) <- c('row1', 'row2', 'row3')
   G
   ```

   (b) Create a diagonal matrix or pull a diagonal out of an existing matrix.

   ```
   # creating diagonal matrix
   diag(1:5)

   # pulling a diagonal out of an existing matrix
   A <- matrix(1:9, nrow = 3)
   diag(A)
   ```

   (c) Create a non-numerical matrix.

   ```
   # matrices don't have to be numerical, but all
      elements are taken to be of the same type
   character.matrix <- matrix(c('this', 'is', 'a',
      'matrix'), nrow = 2, byrow = T)
   character.matrix

   logical.matrix <- matrix(c(T, F, F, F, T, T),
      nrow = 2)
   logical.matrix

   mixed.matrix1 <- matrix(c(1, T, 5, 3), nrow = 2)
   mixed.matrix1 # in this case the resulting
      matrix is taken to be numerical (TRUE = 1)
   ```

```
mixed.matrix2 <- matrix(c(1, T, 'a', 5), nrow =
    2)
mixed.matrix2 # in this case the resulting
    matrix is taken to be string
```

(d) Pull elements out of a matrix.

```
A <- matrix(1:9, nrow = 3)
A[1] # element in the upper right hand corner
A[4] # 4th element of a vector which is contains
     values of all the matrix columns
A[1, 2] # first row, second column
A[1, c(2, 4)] # first row, 2nd and 4th column
A[seq(1, nrow(A), 2), seq(2, ncol(A), 2)] # odd
    rows, even columns

colnames(A) <- c('col1', 'col2', 'col3')
rownames(A) <- c('row1', 'row2', 'row3')
A['row2', 'col3']
A['row2', ]
A[, 'col3']
```

(e) Find out basic information about your matrix such as the number of rows, columns, the value of the determinant etc.

```
dim(A) # dimensions, rows and columns
nrow(A)
ncol(A)
det(A) # determinant
t(A) # transposition
```

10. Get to know a data structure **list.** A list can store different data types or structures in one object.

(a) Create a list combining different variable types.

```
a <- 1:10
b <- c("just", "some", "strings")
M <- matrix(1:15, ncol = 3)

l <- list(a, b, M)
l
```

(b) Name the elements of your list.

```
names(l) <- c("numbers", "words", "matrix")
```

(c) Show information about your list.

```
str(l)
```

(d) Pull elements out of your list.

```
l[[2]] # second element of the list
l$words

l[[2]][1]
l$words[1]
```

11. Get to know a data structure **dataframe**. A dataframe can store numerical, logical and string variables all together in a table format.

(a) Create a dataframe including different types of variables.

```
data.frame(A = 1:3,
           B = c("a", "b", "c"))

df <- data.frame(numbers = 1:10, strings = c('a'
   , 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'
   ), logical = rep(TRUE, 10))
df
```

(b) Pull columns and rows out of a dataframe.

```
df$strings # pull columns by names
df$logical
df[1, 2] # pull values by row and column
df[[2]] # pull the second column
df[, 2] # pull the second column
df[2, ] # pull the second row
df[2, 'logical'] # second element of the '
   logical' column
```

(c) Add columns or rows.

```
df$new.column <- 21:30
df
df[4, ] <- list(11, 'k', TRUE, 31)
df[1, 1] <- 10
df
```

12. Set a working directory from which you want to load data.

```
getwd() # your current working directory
setwd( "insert path here" ) # change working
   directory
```

13. Load a dataset.

```
load("daisy.RData") # load works for RData files
daisy <- read.csv("daisy.csv") # save the read.csv
   output to a variable
# for other data formats use specific libraries

str(daisy)
summary(daisy)
```

14. Check if a column of your data contains an NA value.

```
is.na(daisy$flower.size)
```

15. See the types of variables in your data.

```
head(daisy)
tail(daisy)
```

16. Use some basic functions to see numerical characteristics of your data.

```
# numerical variables
min(daisy$flower.size)
max(daisy$flower.size)
sum(daisy$flower.size)
var(daisy$flower.size)
sd(daisy$flower.size)

# categorical variables
table(daisy$polen)
prop.table(table(daisy$name))
?prop.table
```

17. Create some basic plots to visualise your data.

```
barplot(table(daisy$name))
pie(table(daisy$name))
hist(daisy$flower.size)
# we will see nicer plots in task 23
```

18. See all variables in your workspace. Clear your workspace.

```r
ls()
rm(list = ls())
```

19. Familiarise yourself with loop syntax in R.

```r
a <- 1:10
for (i in 1:length(a)){
  b <- sum(a[1:i])
  print(b)
}

i <- 0
while (i < 3){
  i <- i + 1
  print(i)
}
```

20. Familiarise yourself with if-else conditions in R.

```r
answer <- 42
if(answer == 42){
  print('Correct answer')
} else if(abs(answer - 42) <= 2){
  print('Almost...')
} else {
  print('Wrong answer')
}
```

21. See the syntax for creating functions in R.

```r
add <- function(a, b, c = 1){
    # function adds up given values
  # arguments: a ... number
  # b ... number
  # c ... number (default is 1)
  # returns: number (sum of a, b and c)
  return(a + b + c)
}

add(3, 4)
add(3, 4, c = 2)
```

22. Try installing and using an external library in R.

```r
install.packages('circular')

# load library into current workspace
library(circular)

# execute code from file (useful when you want to
   load external functions)
# source('my-functions.R')
```

23. Get familiar with the basic plot commands in R, try to change the input parametres.

```r
x <- 1:10
y <- 21:30

plot(x, y)
plot(x, y, col = "red")
plot(x, y, col = "red", type = "l")
plot(x, y, col = "red", type = "b")
```

Points:

```r
plot(x, y, col = "red", type = "h")
points(x, y)

# nicer points
plot(x, y, col = "grey30", type = "h", xlab = "X
   label", ylab = "Y label")
points(x, y, pch = 16, cex = 1.25, col = "blue")
```

Lines:

```r
x <- seq(0, 10, by = 0.1)
y1 <- sin(2 * x) / x
y2 <- sin(2 * x) * x / 10

plot(x, y1, col = "darkgreen", type = "l", lwd = 2,
   lty = 3, xlab = "X label", ylab = "Y label")
abline(h = 0, col = "grey60")
lines(x, y2, col = "green")
legend("topright", legend = c("Line 1", "Line 2"),
   col = c("darkgreen", "green"), lty = c(3, 1))
?legend

# we can change limits of y axes
```

```
plot(x, y1, col = "darkgreen", type = "l", lwd = 2,
   lty = 3, xlab = "X label", ylab = "Y label", ylim
    = c(-1, 1))
abline(h = 0, col = "grey60")
lines(x, y2, col = "green")
legend("bottom", legend = c("Line 1", "Line 2"), col
    = c("darkgreen", "green"), lty = c(3, 1), bty =
   "n", horiz = TRUE)
```

Histograms:

```
hist(y2, col = "orange", density = 30, border = "
   orange4", probability = TRUE, xlab = "Data", ylab
    = "Relativ frequencies", main = "Histogram")
box()
# density input specifies the density of shading
   lines
```

Barplots:

```
x <- sample(c("A", "B", "C", "D", "E", "F"), size =
   100, replace = TRUE)
x.tab <- table(x)
barplot(x.tab)

barplot(x.tab, col = 1:6, ylim = c(0, max(x.tab) +
   1), ylab = "Frequencies")
box() # colors indexed by numbers 1:6

barplot(x.tab, col = heat.colors(6), ylim = c(0, max
   (x.tab) + 3), ylab = "Frequencies",
        names.arg = c("A name", "B name", "C name",
           "D name", "E name", "F name"))
box()
text(seq(0.75, 6.75, length = 6), x.tab + 1, labels
   = x.tab)
# colors are from palette heat.colors
# text has inputs: text(x.coordinates, y.coordinates
   , labels)
```