Session 1: Basics, writing scripts, read/write data

Data types

Assigning values with destinctive data types to variables

In order to handle your data right it is very essential to know which type your data is, as different data types have different properties and allow different operations. There are various different data types available in R (e.g. there are several different date formats in R that behave slightly different and are explicitly required for some functions). As a start in R the most essential types to know are:

- **Logical** These only have two states either TRUE or FALSE:

```
# Logical variables
x_true <- TRUE
x_false <- FALSE
```

- **Numeric** These are integer or floating point (double) numbers:

```
# Numeric variables (either integer or double (floating point number)):
x_int <- 1
# if you want to set a variabe explicitly to integer:
x_int <- 1L

x_double <- 1.54
```

- **Character** Text strings of any length. Indicated by apostrophs:

```
# Character variables
x_chr <- "Text"
y_chr <- "Text 1"
z_chr <- "12.426"
```

- **Factor** Very critical data type. It is often mistaken with character or numeric. When it is not explicitly needed in any function (e.g. ANOVA requires factors for the model) factors should be avoided (see data.frames). These are used for categorical data and can only take defined values ("labels") predefined in e.g. a vector. The example below should illustrate the difference between a character vector and the case when the character vector is converted to a factor vector:

```
x_chr <- c("R1", "R2", "R4", "R1", "R1", "R2")
x_fct <- as.factor(x_chr)
x_chr
```

```
## [1] "R1" "R2" "R4" "R1" "R1" "R2"
```

```
x_fct
```

```
## [1] R1 R2 R4 R1 R1 R2
## Levels: R1 R2 R4
```

- **Date formats** As mentioned above there are several date formats available. Here the standard date format is mentioned, as it is the easiest to handle in the beginning. The standard date structure that is automatically recognized by the *as.Date()* function is YYYY-MM-DD. If the date structure in your data differs, you as a user have to give the function the date format as well. This is illustrated in the examples below. How to give the date format to the function can be found in the help of *strptime*:

```
date_1 <- as.Date("2010-12-25")
date_2 <- as.Date("2010-08-05")

# Examples for dates if the date format is not the standard format If the date
# (e.g. in your data) is not given in the standard format as in the two examples
# above, the user has to tell the function as.Date in which format the date is
# given. See the two examples below:
date_3 <- as.Date("25.12.2010", "%d.%m.%Y")

date_chr <- "12:25 05.08.10"
date_4 <- as.Date(date_chr, "%H:%M %d.%m.%y")
```

Querying the data type

To ask which type your data is simply use the function *typeof()*. The outcome for different data types is illustrated below:

```
typeof(x_true)
```

```
## [1] "logical"
```

```
typeof(x_int)
```

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

```
typeof(x_double)
```

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

```
typeof(x_chr)
```

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

```
typeof(x_fct)
```

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

```
typeof(date_1)
```

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

Conversion between different data types

Sometimes it is required to convert your data from one data type to another. Examples are when loading data; R interprets dates in a csv file as characters but the user wants them explicitly as dates, or if for any analysis categorical data is given as numeric or character, but a function requires them explicitly as factors. To convert the data type see following examples:

```
x <- 1.23456
x_chr <- as.character(x)
x_factor <- as.factor(x_chr)
x_logic  <- as.logical(x)
```

Data structures

Overview of the most important data structures

Here I want to show you the most relevant basic data strucures you might face when starting with R. In the overview below I (actually I took this overview from Hadley Wickham's book "Advanced R") ordered them according to the dimensionality of the data and wheter the data structure holds the same data type or also allows different data types in one structure.

Dimension	Homogenious	Heterogenious
1d	Vector	List
2d	Matrix	Data frame
nD	Array	

In simple examples I want to show how to create and subset these data structures.

1d/homogenious: Vectors

All the examples above were actually vectors, in form of single values or the concatination of several values using the operator *c()*. Below you can see different methods to create vectors. Here I want to mention again, that all the values in one vector must be of the same data type. If different data types are mixed they will be converted directly in following order: Logical -> Integer -> Double -> Character

```
# Concatination of values
v_num <- c(1,5,7,2,9)
v_chr <- c("A", "Test", "123", "Last Entry")

# Sequences of values
v_seq <- seq(1,10,1)
v_seq <- 1:10 # if the increment is 1 it can be written in this way as well
v_seq
```

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

```
# Sequence of letters
v_chr <- LETTERS[1:10]
v_chr
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"
```

```
# Repetition of a value n times
v_rep <- rep(1,10)
v_rep
```

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

```
# Repeating the sequence (1 to 10) 10 times
v <- rep(seq(1,10),10)
v <- rep(1:10, 10)

# Subsetting a vector
v[1] # The first value of vector v
```

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

```
v[seq(1,3)] # The first three values
```

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

```
v[c(1,5,8)] # Values at 1, 5, and 8 position
```

```
## [1] 1 5 8
```

```
v[length(v)] # Last value in the vector
```

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

2d/homogenious: Matrix

To define a matrix the user has to provide data that is reshaped in a number of rows and columns that have to be provided by the user. If the provided data is "shorter" than the nrows*ncols then the data is repeated until the matrix is filled. The example below illustrates how to define a matrix:

```
#Defining a matrix with 10 rows and 10 columns with vector v as data input
m_num <- matrix(data = v, nrow = 10, ncol = 10)
```

Subsetting a matrix is similar to subsetting a vector. As a matrix is defined by rows and columns, two indices are required to subset a matrix. The first index always gives the rows and the second the columns.

```
m_num[1:3,1:3]
```

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

```r
m_num[seq(3,1, -1),1] # Reverse order with negative increment in sequence
```

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

1d/heterogenious: List

A list is similar to a vector. Instead of a single value each position in a list can hold any data structure. This means the first position in the list can be a single value, the second can be a whole matrix, and the third can be again a list. The example below is a list with three variables A, B, and C, whereas A and B are vectors and C is a matrix. Furtheron in the example a fourth variable df is added to this list holding a whole data frame (explained below):

```r
# Defining a list
l_1 <- list(A = v_num, B = v, C = m_num)
```

A list is subsetted using the $ operator. as shown in the following example where the variable C is extracted from the list.

```r
# Subsetting a list with the $ operator
l_1$C
```

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

To know how to subset a list is very important, as the output of many analyses are given as lists. To access specific results of your analyses you need the $ operator.

2d/heterogenious: Data frames

The most frequent data structure in R is the data frame, as it is the most intuitive way of providing data. Each column can hold a specific data type (e.g. first column Date, second column categorical data as factors and thrird column holding values as double). Simply spoken, a data frame is a hybrid between a list and a matrix. It can be accessed by its row and column indices. Each column can also be accessed by the $*operator. In the following example a data frame is defined, added to the previously created list and accessed via the *$ operator and indices:

```r
# Defining a data frame
df <- data.frame(numbers = v_seq,
                 letters = v_chr,
                 stringsAsFactors = FALSE)
```

```r
# Example subsetting a more "complex" data strucure -----------------------
# Adding whole data frame as one element in the list
l_1$dataframe <- df

# Fifth element in the column "numbers" of the data frame "df" in the list "l_num"
l_1$dataframe$numbers[5]
```

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

Above you can see that I used the option *stringsAsFactors* = FALSE. This is very important to do when you want to avoid that any text in your data frame should be converted into Factors. This means when you do not explicitly want to work with Factors ALWAYS set this option to FALSE! (also when loading the data e.g. with read.csv)