R Introduction

Help commands

Call help page for functions, data sets etc. using `help()` or `?` as shortcut:

```
help(matrix)
?matrix
```

If the exact name of the, e.g., function is not known, a help search can be performed:

```
help(Matrix)
```

```
## No documentation for 'Matrix' in specified packages and libraries:
## you could try '??Matrix'
```

```
help.search("Matrix")
??Matrix
```

Functions that are not named (e.g., one of the arithmetic operators below) have to be quoted in order to invoke the help function:

```
?"+"
```

Use the hash sign for comments:

```
?+ # error! Use quotation marks!
```

```
## Error: <text>:2:0: unexpected end of input
## 1: ?+ # error! Use quotation marks!
##    ^
```

Everything right of the hash sign is ignored when executing the code.

Basic arithmetics

```
3 + 3      # addition; code works regardless of the spaces, but it is recommended to use them
```

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

```
3 - 5      # subtraction
```

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

```
3 * 5      # multiplication
```

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

```
3 / 5      # division
```

```
## [1] 0.6
```

```
3 ^ 5       # exponentiation
```

```
## [1] 243
```

```
sqrt(81)    # square root
```

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

```r
243 ^ (1/5) # a-th root
```

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

Order of operations:

Multiplication and division first, then addition and subtraction:

```r
1 + 2 * 5 # 1 + 10
```

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

```r
1 - 2 / 5 # 1 - 0.4
```

```
## [1] 0.6
```

The same principle applies to exponentiation and root extraction:

```r
2 * sqrt(9) # 2 * 3
```

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

"." is used as decimal separator:

```r
1,5 + 1,5
```

```
## Error: <text>:1:2: unexpected ','
## 1: 1,
##      ^
```

```r
1.5 + 1.5
```

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

R switches to scientific notation if numbers get too big/too small (according to the value of `scipen` (scientific penalty) that has been set in the global `options()`):

```r
1
```

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

```r
10
```

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

```r
100
```

```
## [1] 100
```

```r
1000
```

```
## [1] 1000
```

```r
10000
```

```
## [1] 10000
```

```r
100000
```

```
## [1] 1e+05
```

```r
0.1
```

```
## [1] 0.1
```

```r
0.01
```

```
## [1] 0.01
```

```
0.001
```

```
## [1] 0.001
```

```
0.0001
```

```
## [1] 1e-04
```

```
?options
options(scipen = 1)
0.0001
```

```
## [1] 0.0001
```

We could also type:

```
1e0
```

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

```
1e5
```

```
## [1] 100000
```

```
1e6
```

```
## [1] 1e+06
```

Objects

Everything in R is an object, all the data, all the functions!

Assignments

```
x <- 5          # store 5 in variable (object) x (assignment)
5 -> x          # also possible
y <- 3          # store 3 in variable y
z <- x + y      # sum of two variables
z <- sum(x, y) # sum of two variables
z
```

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

```
(z <- x - y)    # result of operation is stored in z and printed
```

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

'=' may also be used as assignment operator, but only at the top level, not within the scope of a function; '<-' can be used anywhere -> see help

```
(x <- 5)
```

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

```
(x = 5)  # same result
```

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

```
?'<-'
sum(z = 1:10)
```

```
## [1] 55
```

```
z               # assignment was not made since the scope was within 'sum'
```

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

```
sum(z <- 1:10)
```

```
## [1] 55
```

```
z
```

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

R differentiates between upper- and lowercase letters:

```
Z
```

```
## Error in eval(expr, envir, enclos): object 'Z' not found
```

"A syntactically valid name consists of letters, numbers and the dot or underline characters and starts with a letter or the dot not followed by a number. Names such as '.2way' are not valid, and neither are the reserved words." (See `?make.names`).

Apart from this, there are certain reserved words that cannot be used (see `?Reserved`). To avoid conflicts, you also shouldn't use names of functions and data sets (e.g., `sum()`). If you are not sure if the intended name has already been used, you can check if there is a help page:

```
?s # no documentation found -> o.k.
```

```
## No documentation for 's' in specified packages and libraries:
## you could try '??s'
```

```
?t # function used to transpose a matrix -> not o.k.
```

Functions

R is a functional programming language, and a function call works like this:

```
function.name(argument1 = value1, argument2 = value2)
```

A function is called by its name, followed by brackets containing (zero, one, multiple) function arguments that can be specified. The function arguments are seperated by commas and may or may not have default values. Let us have a look at the `log()` function, used to compute logarithms of numbers:

```
args(log)
```

```
## function (x, base = exp(1))
## NULL
```

It has two arguments, `x` and `base`, but only the latter has a default value. This means that we do have to provide a value for `x`, but not for `base` (i.e., if we do not change the base argument, the natural logarithm will be computed).

```
log(x = 2) # x = 2, base = exp(1)
```

```
## [1] 0.6931472
```

```
log(x = 2, base = 5) # change base
```

```
## [1] 0.4306766
```

```
log(2, 5) # x = 2, base = 5 -> if the argument order is retained, we do not have to name the arguments
```

```
## [1] 0.4306766
```

```
log(5, 2) # x = 5, base = 2
```

```
## [1] 2.321928
```

```
log(base = 5, x = 2) # -> if the arguments are named, it is possible to change their order
```

```
## [1] 0.4306766
```

```
log(b = 5, x = 2) # partial matching -> works as long as the abbreviation of a name is unequivocal
```

```
## [1] 0.4306766
```

Since `x` has no default value, an error occurs if we do not provide a value for it:

```
log()
```

```
## Error in eval(expr, envir, enclos): argument "x" is missing, with no default
```

Needless to say, it is also possible to write one's own functions, although one of R's key advantages is the vast number of ready-made functions covering all types of statistical problems.

Workspace and working directory

```
ls()                    # objects in the workspace
getwd()                 # current working directory
```

Set working directory using slashes (or double backslashes on Windows), but not single backslashes:

```
setwd("/home/mwurzer")  # set working directory using slashes
setwd("C:\\mwurzer")  # ...or double backslashes on Windows, ...
setwd("C:\mwurzer")  # but not single backslashes!
```

Remove objects from workspace:

```
rm(z)
z
```

```
## Error in eval(expr, envir, enclos): object 'z' not found
```

Delete all objects in the workspace:

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

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

Misc.

Nested functions are evaluated from the inside to the outside:

```
y <- 3
(z <- log(exp(y)))
```

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