Swiss Paraplegic Research

Workshop – Introduction into R

R Basics II (Functions)

Yannick Rothacher
Data Scientist

Importing data to R (text files)

- Of course, we generally do not write our data frames by hand but import them from a file (e.g. excel, .txt, .csv, .RData file)
- Excel files can be imported to R but require an additional R-package
- **Text files** (e.g. .csv) are the simplest and most straightforward way to store tabular data
  - Advantage: Are platform independent and can be edited/read using a simple text editor
  - Disadvantage: Can only store variable names and values

- Importing a .csv file (“comma separated values”) to R:

```
> dat <- read.csv("FILENAME.csv")
> dat
  vecA      vecB  vecC
1    2       gut  0.40
2    6 schlecht  0.20
3    7    mittel  0.33
4    9       gut  0.90
```

- For .csv files with a semicolon as separator, we have to specify the separator sign using the **sep** argument:

```
> dat <- read.csv("FILENAME.csv", sep=";")
```

Importing data to R (RData files)

- R-objects can be stored in R-specific .**RData** files (or .**rda,** or .**rdata**)
  - Advantage: Can store multiple R objects
  - Disadvantage: R-specific

- Importing the contents of an .RData file:

```
> load("FILENAME.RData")
```

Working directory

- R always works with reference to a specific location on our computer
- This location is referred to as the "**working directory**"
- If not specified differently, R will always search for files (or save files) in the working directory

- Function to see current working directory:

```
> getwd()
```

- Three ways to set the working directory:

1. Use `setwd()` with the directory path
2. Use the graphical interface of RStudio (Session > Set Working Directory)
3. Start RStudio by opening an R script (working directory will be set to the location of the script)

Closing R

- When terminating R it asks whether the **workspace** should be saved
  - **What this actually means**: R wants to save all created Objects in a **hidden ".RData"** file (files starting with a dot are hidden)
  - Per default when R is started, and it finds such a hidden ".RData" file it will load the content automatically

- We can disable this behaviour in the global options under "Tools > Global Options > General > Workspace"

Functions in R

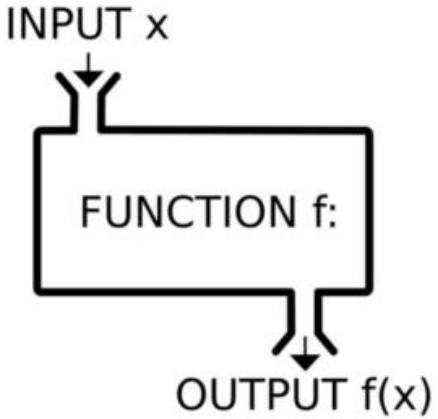
- A function in R classically takes an **input**, processes it and returns an **output**.
- To apply a function, one must write its name followed by normal brackets. Inside the brackets goes the **input** of the function.

- There are many ready-to-use functions in R:
  - Calculate the mean value of a vector:

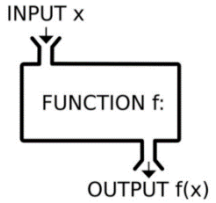
```
> mean( c(10, 15, 20) )
[1] 15
```

  - Return the absolute value of a number:

```
> abs( -2 )
[1] 2
```



Functions in R



- The input to a function can consist of multiple arguments:

```
> sample(x = 1:10, size = 3, replace = TRUE)
[1] 6 3 2
```

- How a function is used (e.g. which input arguments are expected) can be read in the help page of a function. To call the help page we put a question mark before the name of a function:

```
> ?mean
> ?sample
```

- For R-beginners it can be difficult to understand the content of the help pages. As an alternative one always finds helping instructions on the internet.

Functions in R

- Arguments of functions always have **names**. If we do not supply an argument name for an input then the **order of arguments** from the help page will be used for assignment.

```
plot(x, y = NULL, type = "p",  xlim = NULL, ylim = NULL,
     log = "", main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
     ann = par("ann"), axes = TRUE, frame.plot = axes,
     panel.first = NULL, panel.last = NULL, asp = NA,
     xgap.axis = NA, ygap.axis = NA,
     ...)
```

Help page of the plot function (?plot): First comes **x** argument, then **y** argument

```
> plot(x = var1, y = var2)      var1 on x-axis; var2 on y-axis

> plot(y = var2, x = var1)      var1 on x-axis; var2 on y-axis

> plot(var1, var2)              var1 on x-axis; var2 on y-axis

> plot(var2, var1)              var2 on x-axis; var1 on y-axis
```

Installing and using R packages

- An important reason for the popularity of R is the availability of thousands of **additional functions** for all sorts of tasks.

- These functions are available in free **R packages** and can be installed from the Comprehensive R Archive Network (CRAN)

- R packages are developed and maintained by R users. Everybody can write and publish an R package.

Installing and using R packages

- A new R package must first be installed:
  ```
  > install.packages("PACKAGENAME")
  ```
  (can also use the graphical interface of RStudio)

- The installation has to be done only once.

- To directly use the contents of a package, we first must load it:
  ```
  > library(PACKAGENAME)
  ```

- The loading of a package must be done again in each new R session.

Creating your own function

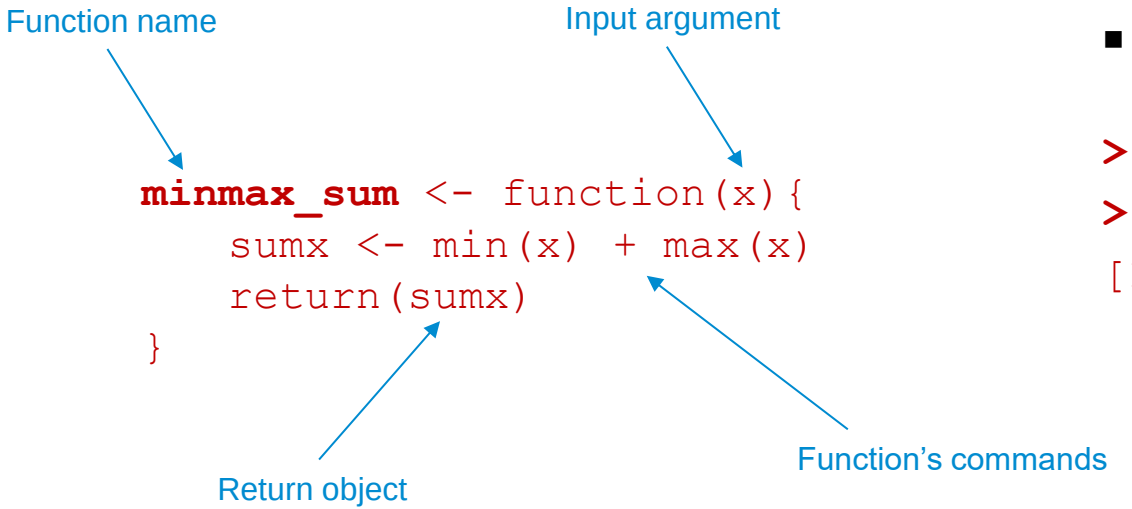
- Creating a function in R follows the syntax:

```
function_name <- function(arg1, arg2, ...){
     Function body
     return(return_object)
}
```

- The function_name is the name under which the function is stored
- The arguments (arg1, …) define the input objects
- The Function body are the commands that the function executes
- Classically a function contains a return object which is the output of the function

Creating your own function

- Simple example of a function
  - Calculate the sum of the smallest and largest element of a (numeric) vector:



- Apply the function:

```
> vec <- 1:20
> minmax_sum(x = vec)
[1] 21
```

Creating your own function

- A function can take **more than one** input argument
  - Function calculating the sum of the smallest element of one vector and largest element of another vector:

```
minmax_sum <- function(x, y){
    sumxy <- min(x) + max(y)
    return(sumxy)
}
```

  - Apply the function:

```
> vec1 <- 1:20
> vec2 <- 10:15
> minmax_sum(x=vec1, y=vec2)
[1] 16
```

  - If we do not specify the input, R returns an error:

```
> minmax_sum(x = vec1)
Error in minmax_sum(x = vec1) :
argument "y" is missing, with
no default
```

Creating your own function

- We can define default values for the input arguments:

```
minmax_sum <- function(x, y = 15:20){
    sumxy <- min(x) + max(y)
    return(sumxy)
}
```

- Apply the function:

```
> vec1 <- 1:3
> minmax_sum(x=vec1)
[1] 21
```

- The return object can be any R-object (e.g. number, vector, data frame…)
- What if we want to produce **multiple output objects**?
  - In this case we need to use a **list** as a return object

Lists in R

- A **list** is like a vector but each element is an independent R-object
- For example: A list can store a **numeric vector** as a first element, **a data frame** as a second element and **a character vector** as a third element:

```
> vec <- 1:5
> d <- data.frame(col1=1:3, col2=11:13)
> char <- c("a", "b", "c")

> mylist <- list(A=vec, B=d, C=char)
```

```
> mylist
$A
[1] 1 2 3 4 5

$B
  col1   col2
1    1       11
2    2       12
3    3       13

$C
[1] "a" "b" "c"
```

Lists in R

- To access individual elements of a list use double square brackets [[ ]]:

```
> mylist[[1]]
[1] 1 2 3 4 5

> mylist[[2]]
  col1 col2
1    1   11
2    2   12
3    3   13
```

```
> mylist
$A
[1] 1 2 3 4 5

$B
  col1 col2
1    1   11
2    2   12
3    3   13

$C
[1] "a" "b" "c"
```

List as a function output

- By using a list as a return object, we can include multiple objects in the output:

```
> minmax_sum <- function(x, y){
    sumxy <- min(x) + max(y)
    listout <- list(sum=sumxy,
                    input1=x,
                    input2=y)
    return(listout)
  }
```

```
> res <- minmax_sum(x = 1:5,
                    y = c(99,3))
> res
$sum
[1] 100

$input1
[1] 1 2 3 4 5

$input2
[1] 99 3
```

Exercise: R functions

- Find the exercise at:

https://github.com/Swiss-Paraplegic-Research/Workshop/tree/main/**Part2_RFunctions/Exercise**

EXTRA: Working directory

- On Windows, finding a file path and using it in R can be cumbersome because Windows uses the backward slash (\) as a directory separator whereas R uses the more universally accepted forward slash (/)
- R actually lets us use the backward slash as well, but because the backward slash has a different meaning in R, we must “escape” it by writing two slashes:
  `"C:\\Users\\rothacher_y\\Desktop"`

- One alternative:
  1. Copy the path in Windows (e.g. right-click on file and copy “location”)
  2. Use the readClipboard() function to automatically add double slashes
     ```
     > setwd(readClipboard())
     ```

EXTRA: Scope of a function

- Variables created inside a function are not accessible afterwards:

```
> minmax_sum <- function(x, y){
    sumxy <- min(x) + max(y)
    listout <- list(sum=sumxy,
                    input1=x,
                    input2=y)
    return(listout)
  }
```

```
> res <- minmax_sum(x = 1:5,
                    y = c(99,3))
> listout
Error: object 'listout' not found
```

- The reason for this behaviour lies in the way R stores and searches for objects.
- Function commands are executed in a separate environment which is deleted after the function has been executed.