

Programmieren mit R für Einsteiger



Berry Boessenkool



frei verwenden, zitieren

2022-03-09 14:31

0. Intro




1. Grundlagen

0.1 Willkommen

0.2 Fünf-Minuten Showcase

0.3 Einrichtung: R & Rstudio

0.4 Übungsaufgaben

- ▶ **Berry:** Geoökologie @ Uni Potsdam, Dozent am HPI
 - ▶ R Fan seit 2010
 - ▶ R-Pakete  , Training & Beratung  , Community 
-

- ▶ **Pia:** Medizin @ University of St Andrews, Edinburgh Scotland
 - ▶ Master Digital Health am HPI (Digital Health Center, DHC)
 - ▶ RKI Intensivregister: Grafiken für Kristenstäbe / Bürger*innen
-

- ▶ **Bert:** Leiter Lehrstuhl Digital Health - Connected Healthcare (HPI)
- ▶ Studiengangsverantwortlicher Masterprogramm Digital Health
- ▶ R für Analyse von gesundheits-relevanten Daten aus dem Alltag

- ▶ Grundlagen und Anwendungen (möglichst konkret)
- ▶ Freude am Programmieren
- ▶ Gelerntes in eigenen Projekten verwenden
- ▶ effiziente und reproduzierbare Arbeit
- ▶ Inhaltlicher Ablauf auf openHPI
- ▶ Video pro Lektion
- ▶ Übungsaufgaben auf CodeOcean (im Browser) / in Rstudio
- ▶ Fragen? -> **Kursforum!** [Diskussionen] (unter den Videos erstellen)

0. Intro

1. Grundlagen

0.1 Willkommen

0.2 Fünf-Minuten Showcase

0.3 Einrichtung: R & Rstudio

0.4 Übungsaufgaben

- ▶ 5 Minuten Beispiel für die Nutzung von R und Rstudio
- ▶ Als 'Live Coding' gezeigt, d.h. der Code wird im Skript entwickelt und vorgeführt
- ▶ Die nachfolgenden Folie zeigt den Code zur Referenz

- ▶ Ziel ist nicht, alles im Detail zu verstehen,
- ▶ sondern 'Appetit' auf R zu bekommen :)
- ▶ Am Kursende kannst du diese Sachen auch

- ▶ sit back, relax, enjoy the show...

```
# R als Taschenrechner: -> 'sinnvolles' Ergebnis
7 * 6

# Objekte erstellen um später damit zu arbeiten:
alter <- 33
alter + 1      # so alt bin ich nächstes Jahr
2022 - alter   # so finde ich raus, wann ich geboren wurde, falls ich das mal vergesse

# Daten einlesen:
wetter <- read.table(file="wetter.txt", header=TRUE)

# Aus einer Tabelle eine Spalte auswählen:
wetter$Regen

# Graphik - Scatterplot für Korrelation (Zusammenhang, kann aber dritte Gründe haben):
plot(wetter$Sonne, wetter$Temperatur, col="orange", pch=16, main="Viel Sonne an warmen Tagen")

# Graphik - Zeitreihe für Entwicklung / Saisonalität / Trend:
wetter$Datum <- as.Date(wetter$Datum, format="%Y-%m-%d")
plot(wetter$Datum, wetter$Luftdruck, type="l", col="salmon", lwd=2, xaxt="n")

# Pakete von anderen R Nutzern:
library(berryFunctions)
monthAxis()
```

0. Intro

1. Grundlagen

0.1 Willkommen

0.2 Fünf-Minuten Showcase

0.3 Einrichtung: R & Rstudio

0.4 Übungsaufgaben



- ▶ Programmiersprache für Datenanalyse / -visualisierung und Statistik in Forschung und Wirtschaft
- ▶ kostenlos, open source (nachvollziehbar, erweiterbar)
- ▶ große Nutzer-community (viele Methoden)
- ▶ macht deine Arbeit effizient und produktiv



- ▶ **I**ntegrated **D**evelopment **E**nvironment (IDE) für R:
Umgebung zur Entwicklung von Code, mit R integriert

Installation

- ▶ **Anleitung**
- ▶ Für den Kurs bitte eine aktuelle R **V**ersion (> 4.0) nutzen

The image shows the RStudio desktop environment. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, and Help. Below the menu is a toolbar with icons for file operations and running code. The main editor window on the left displays an R script with the following code:

```
1 # R in 5 minutes
2 # Berry Boessenkool
3
4 # R as calculator:
5 7 * 6
6
7 # create objects for later use
8 height <- 183
9 height - 5
10
11
12
13
14
15
16
```

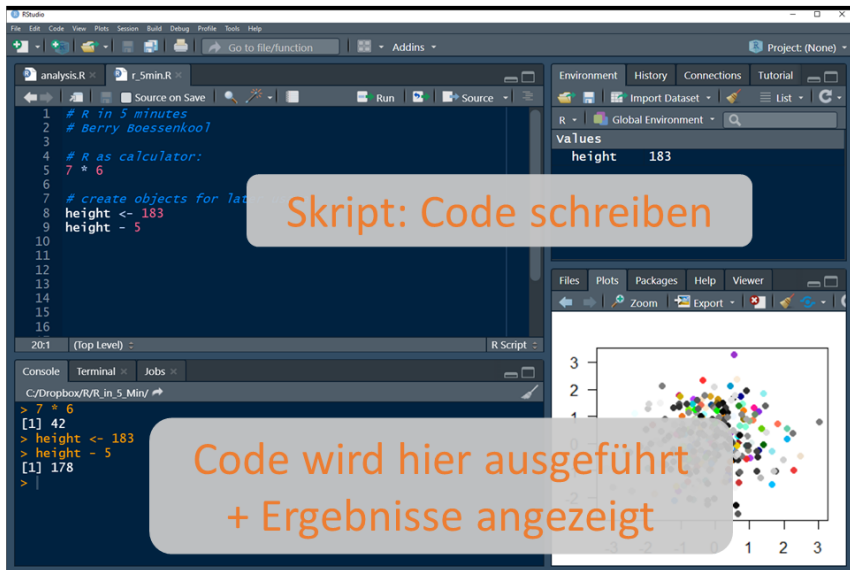
A callout box labeled "Skripte (.R Dateien)" points to this script editor. The right-hand pane shows the Environment tab with a table of objects:

| values | |
|--------|-----|
| height | 183 |

A callout box labeled "Objekte" points to this Environment pane. The bottom-left pane shows the Console with the following output:

```
> 7 * 6
[1] 42
> height <- 183
> height - 5
[1] 178
> |
```

A callout box labeled "Die Konsole zum tatsächlichen R" points to this console. The bottom-right pane shows a scatter plot with data points colored by density. A callout box labeled "Grafiken" points to this plot pane.



The screenshot displays the RStudio environment. The top-left pane shows a script file named `r_5min.R` with the following code:

```
1 # R in 5 minutes
2 # Berry Boessenkool
3
4 # R as calculator:
5 7 * 6
6
7 # create objects for later use
8 height <- 183
9 height - 5
```

The top-right pane shows the Environment window with the variable `height` assigned the value 183.

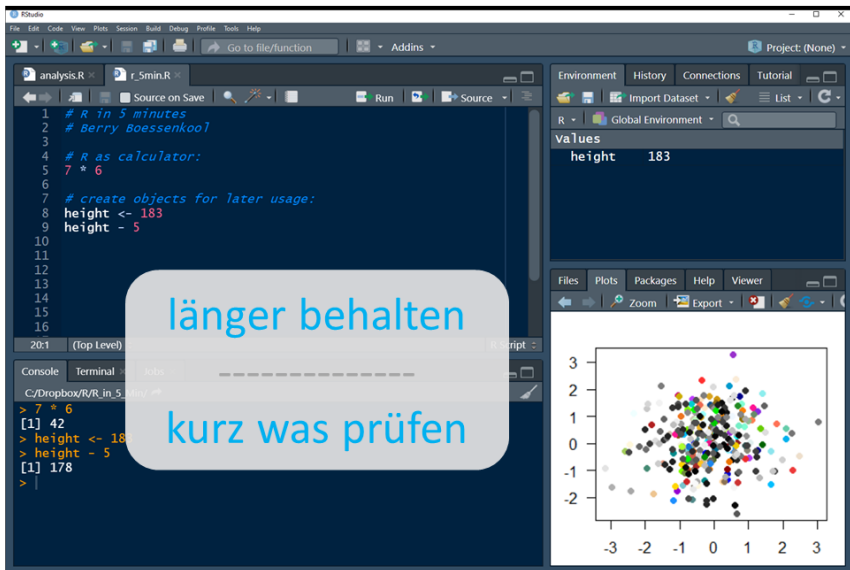
The bottom-left pane shows the Console window with the following output:

```
> 7 * 6
[1] 42
> height <- 183
> height - 5
[1] 178
> |
```

The bottom-right pane shows a scatter plot of data points, with axes ranging from -3 to 3 on both the x and y dimensions.

Two text overlays are present:

- Skript: Code schreiben** (Script: Write code) is positioned over the script editor.
- Code wird hier ausgeführt + Ergebnisse angezeigt** (Code is executed here + results are displayed) is positioned over the console window.

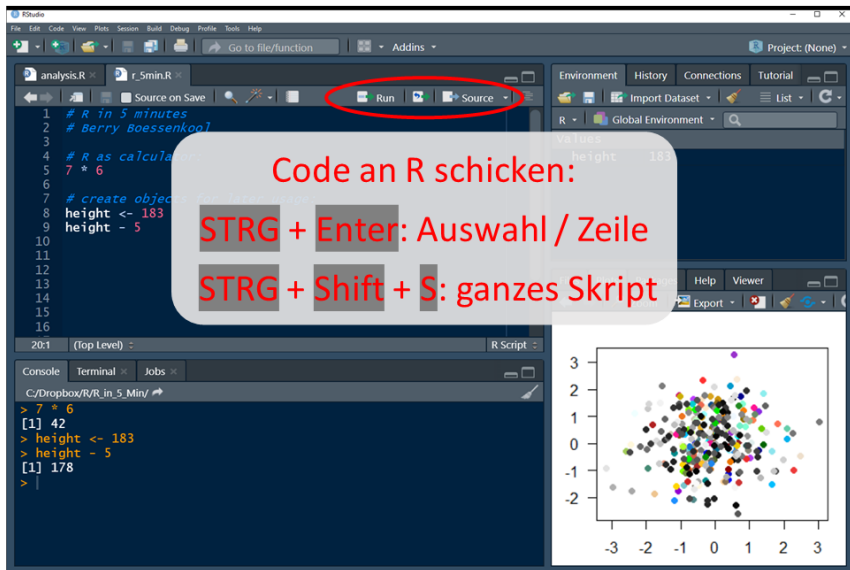


The screenshot shows the RStudio IDE with the following components:

- Source Editor:** Contains R code for a script named `r_5min.R`. The code includes comments and assignments for a variable `height`.
- Environment:** Shows the `Global Environment` with a single variable `height` having a value of `183`.
- Console:** Displays the output of the code execution, showing the result of the multiplication and the assignment of `height`.
- Plots:** A scatter plot is displayed in the bottom right pane, showing a distribution of data points.

Overlaid on the console area are two text boxes:

- A light blue box with the text `länger behalten` (keep longer).
- A light blue box with the text `kurz was prüfen` (check briefly).



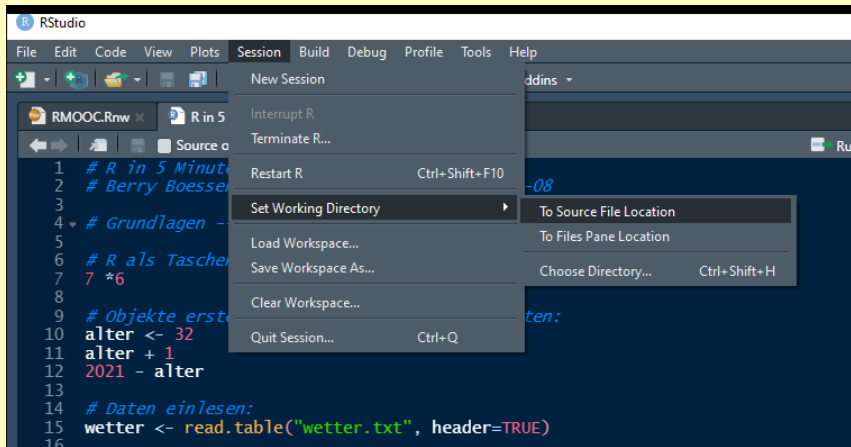
The screenshot displays the RStudio environment. The source editor on the left contains R code for a script named 'r_5min.R'. The code includes comments and assignments: `height <- 183` and `height - 5`. The Run button in the top toolbar is circled in red. A semi-transparent text box in the center provides instructions: 'Code an R schicken: STRG + Enter: Auswahl / Zeile' and 'STRG + Shift + S: ganzes Skript'. The console at the bottom left shows the output of the executed code: `[1] 42` and `[1] 178`. The environment pane on the right shows the 'Global Environment' with a variable 'height' having a value of 183. A scatter plot is visible in the bottom right corner, showing a distribution of points.

Code an R schicken:
STRG + Enter: Auswahl / Zeile
STRG + Shift + S: ganzes Skript

Einrichtung der R Arbeitsumgebung:

- ▶ R & Rstudio installieren und nutzen
- ▶ Skripte
- ▶ Zeilen ausführen

Für einfache Skripte:



The image shows a sequence of four screenshots illustrating the steps to create a new R project in RStudio:

- RStudio File Menu:** The 'File' menu is open, and 'New Project...' is highlighted.
- New Project Wizard - Create Project:** The 'Create Project' screen shows three options: 'New Directory' (Start a project in a brand new working directory), 'Existing Directory' (Associate a project with an existing working directory), and 'Version Control' (Checkout a project from a version control repository). The 'New Directory' option is selected.
- New Project Wizard - Project Type:** The 'Project Type' screen shows a list of project types. 'New Project' is selected, and a tooltip indicates 'Create a new project in an empty directory'.
- New Project Wizard - Create Project from Existing Directory:** The 'Create Project from Existing Directory' screen shows the 'Project working directory' field with the path 'C:/Users/berry/Desktop/HP_pdf' entered. The 'Browse...' button is visible next to the field.

Soll deine Arbeit reproduzierbar sein, setze unter

Rstudio - Tools - Global Options - General

OFF: Restore .Rdata into workspace at startup

Save workspace to .RData on exit: **NEVER**

Weitere hilfreiche **Rstudio Einstellungen**

Im Folgenden einige praktische Tastaturkürzel aus

rviews.rstudio.com/categories/tips-and-tricks

Rstudio keyboard shortcuts (**ALT** + **SHIFT** + **K**)**STRG** + **ENTER** im Skript: Zeile / Auswahl an R senden**UP** / **DOWN** in der Console: letzte Befehle**STRG** + **UP** Befehlsgeschichte**STRG** + **SHIFT** + **P** vorherige Auswahl (mit Änderungen) senden**STRG** + **SHIFT** + **S** / **ENTER** Gesamtes Skript ausführen**STRG** + **SHIFT** + **N** Neue Skriptdatei anlegen**STRG** + **O** / **S** Datei öffnen / Skript speichern**STRG** (+ **SHIFT**) + **TAB** nächstes (vorheriges) Skript**ALT** + Maus für Mehrzeilen-Cursor. *oder:* **STRG** + **ALT** + **UP** / **DOWN**

Windows Bildschirmrotation ausschalten: Desktop Rechtsklick - Grafikoptionen - Tastenkombinationen - deaktivieren

ALT + **UP** / **DOWN** Zeile im Skript verschieben**STRG** + **1** Cursor auf Panel (1:source, 2:console, 3:help, 4:history, 5:files, 6:plot...)**STRG** + **SHIFT** + **1** / **2** / **3** / ... Panel Vollbild**STRG** + **SHIFT** + **C** Zeile auskommentieren**STRG** + **SHIFT** + **O** Inhaltsübersicht (**# abschnitt ----**)**STRG** + **SHIFT** + **M** pipe operator einfügen

- ▶ Working directory (Arbeitsverzeichnis)
- ▶ Projekte
- ▶ Effektiv arbeiten in Rstudio

0. Intro

1. Grundlagen

0.1 Willkommen

0.2 Fünf-Minuten Showcase

0.3 Einrichtung: R & Rstudio

0.4 Übungsaufgaben



Zu jeder Lektion in diesem Kurs gibt es interaktive Code-übungen ($n=25$), die aus 5 - 15 Aufgaben mit steigender Komplexität bestehen. Sie sind über die openHPI-Plattform zugänglich (nicht per URL direkt, da der Login über openHPI erfolgt).


Exercise 1

[Edit item](#) [Statistics](#)

Instructions:

Click the button below to launch the exercise.

 This is a graded exercise.  10.0 points

 Launch exercise tool

Sie können im Browser gelöst werden oder (besser): heruntergeladen und in Rstudio gelöst werden.

Für jede Woche gibt es eine unbewertete Spielwiese mit dem Code der Folien.

Übungen auf CodeOcean - ausführen

Sobald die Aufgabe geöffnet ist, kannst du im Skript schreiben und es ausführen durch Klicken auf 'Ausführen' (**ALT** + **R**)

The screenshot displays the CodeOcean web interface. At the top, the browser address bar shows the URL `https://codeocean.openhpi.de/exercises/721/implement`. The CodeOcean header includes the logo, 'Administration' link, 'English' language selector, 'Help' link, and the user 'Berry Boessenkool'. Below the header, a breadcrumb trail reads 'EXERCISES / R EXERCISE MASTER TEMPLATE / IMPLEMENT'. The main content area is titled 'R exercise master template' with a '0%' progress indicator and a '(Show)' link. A tooltip indicates the keyboard shortcut 'ALT + R' for the 'Run' button. The interface is divided into several sections: a left sidebar with 'Collapse Action Sidebar' and a 'Files' list containing 'examples_1.R', 'examples_2.R', and 't_dataset.txt'; a central editor with a 'Run' button, 'Score' tab, and 'Request Comments' button, displaying an R script; and a right sidebar with 'Collapse Output Sidebar' showing the output of the script. The R script in the editor includes comments and code for creating objects and running a function. The output on the right shows the execution of `codeoceanR::rt_score()` resulting in a vector of integers from 1 to 15.

```
# Structure of task files
# Task 1 -----
# Create an object named 'my_first_object' with the
# number 99.
# Make sure to save the script (CTRL + S) before running
# the following:
codeoceanR::rt_score()
# You can conveniently save + source (= run full script
# including previous line)
# by clicking on "Source" (topright if script window) or
# pressing CTRL + SHIFT + S.

# Task 2 -----
# Create another object with the integer values from 5
# till 15.
# The desired objectnames is already included for your
# convenience.
# Replace the zero with the intended code for the
# solution.
my_second_object <- 5:15
my_second_object

# Now continue in examples_2.R
```

```
> codeoceanR::rt_score()
NULL
> my_second_object <- 5:15
> my_second_object
[1] 5 6 7 8 9 10 11 12 13 14 15
```

Übungen auf CodeOcean - bewerten

So oft du willst kannst du deine Lösungen überprüfen lassen durch Klicken auf 'Bewerten' (**ALT** + **S**)

ster template

Keyboard shortcut: ALT + s

100%

Run

Score

Request Comments

Collapse Output Sidebar

```

1 # Structure of task files
2
3 # Task 1 -----
4 # Create an object named 'my_first_object' with the
  number 99.
5 my_first_object <- 99
6
7 # Make sure to save the script (CTRL + S) before running
  the following:
8 codeoceanR::rt_score()
9 # You can conveniently save + source (= run full script
  including previous line)
10 # by clicking on "Source" (topright if script window) or
  pressing CTRL + SHIFT + S.
11
12
13
14 # Task 2 -----
15 # Create another object with the integer values from 5
  till 15.
16 # The desired objectnames is already included for your
  convenience.
17 # Replace the zero with the intended code for the
  solution.
18 my_second_object <- 5:15
19
20
21 # Now continue in examples_2.R
22

```

Results

1 test files have been executed.

Test File 1 (examples_tests.R)

| | |
|-----------------------|--|
| Passed Tests | 8 out of 8 |
| Score | 8 out of 8 |
| Feedback | Well done. All tests have been passed. |
| Error Messages | |

Score: 100%

Bewerte oft, da die Meldungen immer spezifischer werden, je näher du der gewünschten Lösung kommst. Übertrage am Ende den Punktestand an openHPI durch Klicken auf "Code zur Bewertung abgeben".

Du kannst die Übungen in Rstudio lösen und dadurch in der normalen R-Arbeitsumgebung arbeiten, eine Zeile / Auswahl von Code ausführen, die Autovervollständigung nutzen, teilweise offline arbeiten (außer beim Bewerten), Tastaturkürzel verwenden, Debugging-Tools genießen, integrierte Grafikausgaben sowie Hilfe, Paketverwaltung, Versionskontrolle und mehr erhalten.

Hierfür musst du unser Paket installieren. Dazu Folgendes kopieren und ausführen:

```
install.packages("remotes")  
remotes::install_github("openHPI/codeoceanR")
```

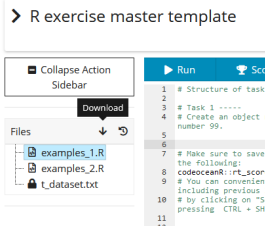
Dies ist nur einmal erforderlich und sollte weniger als eine Minute dauern.

Auf Linux zuerst `curl` und `openssl` installieren, siehe [Hinweise](#) (ggf. Anleitung befolgen, z.B. `sudo apt install libcurl4-openssl-dev`):

```
install.packages("curl")  
install.packages("openssl")
```


Für jede Übung:

- ▶ 1. via OpenHPI die CodeOcean Übung öffnen
- ▶ 2. downloaden, in geeignetem Ordner auf dem Rechner speichern (entpacken optional)



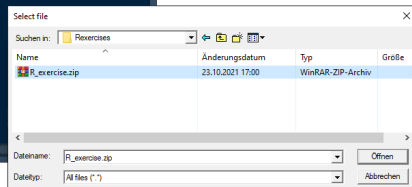
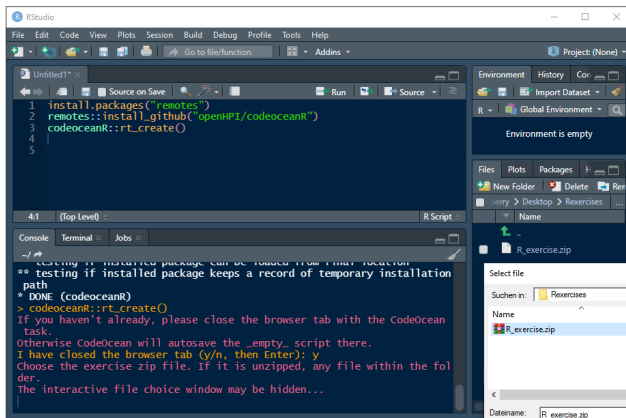
Gelegentlich meldet CodeOcean den Fehler "Sorry, something went wrong".
-> Ignorieren, wenn der Download beginnt. Andernfalls Seite neu laden.

- ▶ 3. Die CodeOcean Registerkarte im Browser schließen (damit CO das dortige ungelöste Skript nicht automatisch speichert)

- ▶ 4. folgendes in R / Rstudio ausführen:

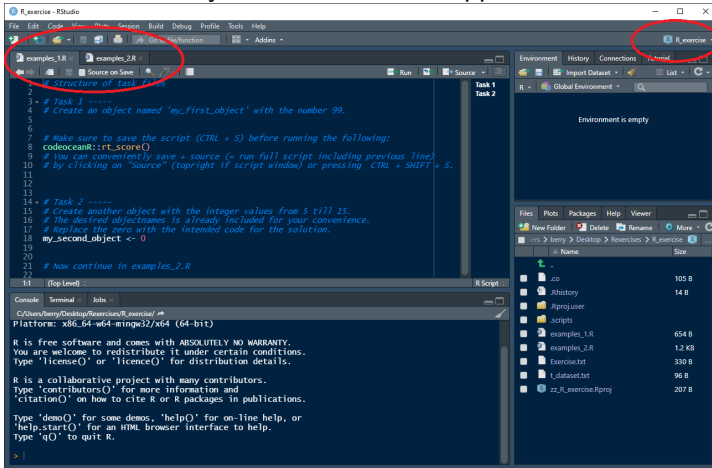
```
codeoceanR::rt_create()
```

- ▶ - bestätigen, den Browser Tab geschlossen zu haben
- ▶ - Datei auswählen (wenn entpackt, irgendeine Datei innerhalb des Ordners)



Übungen in Rstudio - Aufgaben bearbeiten

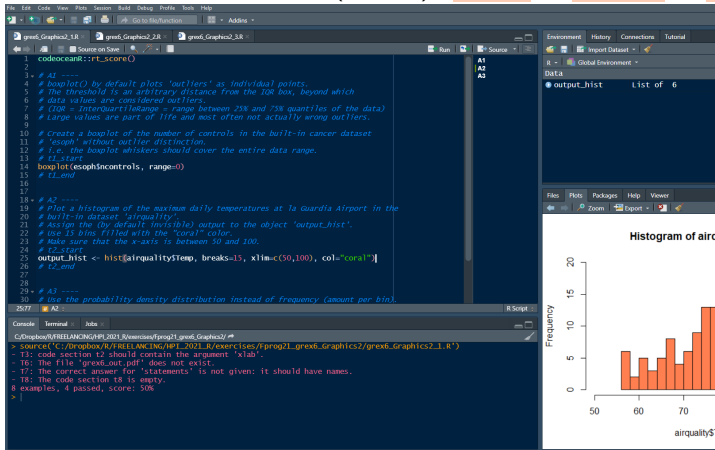
`rt_create` sollte ein neues Rprojekt in einer separaten Rstudio-Instanz öffnen mit bereits geöffneten Übungsskriptdateien.
Kontaktiere Berry, wenn das nicht klappt.



Die Übung kann jederzeit mit der `zz_*.Rproj` Datei geöffnet werden.

codeoceanR: :rt_score()

sendet deinen Code an CodeOcean (sichtbar bei erneutem Öffnen im Browser), führt das Testskript aus und zeigt die Rückmeldungen in Rstudio. Ganzes Skript ausführen (source): **CTRL** + **SHIFT** + **S**



interaktive Übungsaufgaben

- ▶ auf CodeOcean
- ▶ mit verstecktem Skript zum Testen deiner Lösungen
- ▶ in Rstudio: Übung runterladen, Registerkarte schließen, `codeoceanR::rt_create()` ausführen
- ▶ mehrmals Punkte testen lassen (Score), nur einmal einreichen (Submit)
- ▶ Scoring-Meldungen werden zunehmend spezifischer

0. Intro

1. Grundlagen

1.1 Syntax

1.2 Hilfe

1.3 Vektoren

1.4 Statistik

```
4*9
```

```
## [1] 36
```

Code in diesen Folien ist mit grauen Kästen hinterlegt und farbig markiert (syntax-highlighting). R Ausgaben stehen hinter `##`.

`[1]` wird im Abschnitt 2.3 (Vektoren) erklärt.

Kommentare (nach einem Hashtag) werden von R ignoriert.
Sie machen Code für Menschen verständlich = leichter lesbar:

```
2 + 4.8 # Punkt als Dezimaltrennzeichen
```

```
## [1] 6.8
```

```
2+4.8 # Leerzeichen egal für R, hilfreich für Lesbarkeit
```

```
3^2
```

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

```
sqrt(81) # Wurzel (square root)  
## [1] 9
```

```
abs(-12) # Betrag (absolute value)  
## [1] 12
```

```
log(100) # natürlicher Logarithmus (ln) mit Basis e (2.72)  
## [1] 4.60517
```

```
log10(100) # Logarithmus mit Basis 10  
## [1] 2
```

```
exp(3) # Exponentialfunktion  $e^3$   
## [1] 20.08554
```


Zuweisung (assignment): ein Objekt mit Daten im Workspace anlegen

```
alter <- 15.4
```

Rstudio: Tastaturtasten **ALT** + **-** drücken für **<-**

```
alter # ist jetzt im Workspace (quasi der R Speicher)  
## [1] 15.4
```

```
alter + 5  
## [1] 20.4
```

alter ist unverändert 15.4. Zum Ändern überschreiben:

```
alter <- 37.1  
alter # immer die aktuelle Version, keine Geschichte dabei  
## [1] 37.1
```

Groß-/Kleinschreibung (case) beachten:

```
Alter # ist kein existierendes Object  
## Fehler: Objekt 'Alter' nicht gefunden
```

```
ls() # Eigene Objekte im Workspace auflisten
```

```
## [1] "alter"
```

```
rm(alter) # ein Objekt löschen
```

```
pi # Eingebaute Konstante
```

```
## [1] 3.141593
```

```
pi <- 3 # das kann überschrieben werden
```

```
sin(pi/2) # aber dann kommt nicht mehr ganz 1 raus ...
```

```
## [1] 0.997495
```

Empfehlung: bestehende Namen wie `pi`, `sin` nicht verwenden. Wenn ein eigenes Objekt `pi` existiert, wird nicht das eingebaute `pi` genutzt.

Gute Objektamen sind **kurz aber aussagekräftig**, zB `tempMaxBerlin`

Übliche Konventionen sind `lowerCamelStandard` und `unter_strich`.

Den alten `punkt.standard` bitte nicht mehr verwenden. Der hat in anderen Programmiersprachen eine besondere Bedeutung.

Funktionen werden mit runden Klammern aufgerufen (ausgeführt):

```
log(7.4) # Funktionsaufruf  
## [1] 2.00148
```

```
log(x=7.4) # explizite Benennung des Arguments  
## [1] 2.00148
```

Argumente haben Namen. Diese können weggelassen werden, sofern sie in der richtigen Reihenfolge stehen.

`log` hat ein weiteres Argument `base`. Wenn das (wie bisher) nicht angegeben wird, wird 2.718 verwendet. Das ist der Standardwert für `base` (default). Für benutzerdefinierte Basis:

```
log(x=200, base=12)  
## [1] 2.1322
```

Argumentnamen können abgekürzt werden, sofern sie einmalig sind:

```
log(200, b=12)  
## [1] 2.1322
```

Syntax, Objekte, Operatoren, Funktionen:

- ▶ `+`, `-`, `*`, `/`, `^`
- ▶ Leerzeichen und Kommentare (`#`) machen Code leichter lesbar
- ▶ `pi`, `sqrt`, `abs`, `log`, `log10`, `exp`
- ▶ Objekte: `ALT` + `-` für Zuweisungsfeil (`<-`)
- ▶ Objektnamen Case sensitive
- ▶ `ob_jekt` oder `objekt`, nicht: `pi`, `daten`,
- ▶ `ls`, `rm`
- ▶ `funktion(1, argument=2, arg=3)`

Operatoren 2.0 ; Exponentialdarstellung

```
sin(15 * pi/180) # Grad zu dezimal  
## [1] 0.247404
```

```
factorial(5) # Fakultät:  $n! = 1*2*3*4*...*n$   
## [1] 120
```

```
exp(1) # eulersche Zahl e  
## [1] 2.718282
```

```
e^3 # geht so nicht  
## Fehler: Objekt 'e' nicht gefunden
```

```
3.91 * 10^-3 # wissenschaftliche Schreibweise: 3,91 E-3  
## [1] 0.00391
```

```
3.91e-3 # scientific notation: keine Leerzeichen erlaubt  
## [1] 0.00391
```

```
1e6 # schnell eine Million schreiben (ohne 6 Nullen)  
## [1] 1e+06
```

```
options(scipen=9) # bis 1e9 wird ab jetzt ausgeschrieben  
1e6 ; 1e14  
## [1] 1000000  
## [1] 1e+14
```

Technisch kann `=` statt `<-` für Zuweisungen verwendet werden. Gemäß **style guide** sollte `=` nur für `Funktion(arg="wert")` verwendet werden.

`median(x <- 1:10)` erstellt auch `x` im `globalenv()` workspace, `median(x=1:10)` nicht.

blog.revolutionanalytics.com, csgillespie.wordpress.com

0. Intro

1. Grundlagen

1.1 Syntax

1.2 Hilfe

1.3 Vektoren

1.4 Statistik

Dokumentation eingebauter Funktionen

`help("append")` # Doku der Funktion 'append' öffnen
`?append` # Schnellvariante um weniger zu tippen

Noch schneller: **F1** drücken, während der Cursor auf dem Befehl ist

(**Fn** bei Laptops)

Paket, in dem die Funktion enthalten ist

Titel & Beschreibung,
 manchmal zusätzliche Info
 im Abschnitt **Details** oder **Note**

Argumente, ggf. mit Defaults (Standardwerte)

Beschreibung der Argumente

Ausgabe der Funktion (zurückgegebener Wert)

Verweise, oft auch unter **See Also**

Beispiele, oft sehr hilfreich!

Weitere Info zum Paket

append (base) R Documentation

Vector Merging

Description
 Add elements to a vector.

Usage
`append(x, values, after = length(x))`

Arguments

`x` the vector the values are to be appended to.
`values` to be included in the modified vector.
`after` a subscript, after which the values are to be appended.

Value
 A vector containing the values in `x` with the elements of `values` appended after the specified element of `x`.

References
 Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Examples
`append(1:5, 0:1, after = 3)`

[Package base version 4.0.3 [Index](#)]


```
help.search("append") # Alle verfügbaren Doks mit Bezug
??append # auch hier wieder eine kürzere Variante
```

```
help.start() # offline Handbücher und Material
```

- ▶ **Kursforum** für dieses MOOC
- ▶ **StackOverflow** für Programmierfragen <– **Wichtigste Ressource**
- ▶ (online) Buch: Golemund & Wickham (2017) - **R for Data Science**
- ▶ Deutsches Buch: Uwe Ligges (2005) - Programmieren mit R
- ▶ **Reference Card**: Tom Short & Jonas Stein (2013)
- ▶ **base** und **advanced** Cheatsheets von **Rstudio**
- ▶ Mehr unter bookdown.org/brry/course/resources
- ▶ Online R Instanzen: rdrr.io/snippets, cocalc.com, colab.to/r

Antworten auf R Fragen finden:

- ▶ Dokumentationen von Funktionen aufrufen (? / F1)
- ▶ Stackoverflow
- ▶ Kursforum
- ▶ RefCard & Bücher

0. Intro

1. Grundlagen

1.1 Syntax

1.2 Hilfe

1.3 Vektoren

1.4 Statistik

Vektoren in R sind keine geometrischen Konstrukte, sondern eine geordnete Menge. (ordered set of values).

Vektoren werden erstellt mit `c` (Combine / Concatenate). Einträge werden mit einem Komma getrennt.

```
zahlen <- c(3, 7, -2.7654321, 11, 3.8, 9)
```

Objekt aufrufen (anzeigen):

```
zahlen
## [1]  3.000000  7.000000 -2.765432 11.000000  3.800000
## [6]  9.000000
```

```
print(zahlen, digits=3) # Explizit anzeigen, mit Optionen
## [1]  3.00  7.00 -2.77 11.00  3.80  9.00
```

```
1:5 # Ganze Zahlen (integers) von : bis
## [1] 1 2 3 4 5
```

```
rep(1:4, times=3) # Zahlen mehrfach wiederholen
## [1] 1 2 3 4 1 2 3 4 1 2 3 4
```

```
rep(1:3, each=3, times=2)
## [1] 1 1 1 2 2 2 3 3 3 1 1 1 2 2 2 3 3 3
```

```
seq(from=3, to=-1, by=-0.5) # Sequenz
# Für absteigende Folgen muss 'by' negativ sein
## [1] 3.0 2.5 2.0 1.5 1.0 0.5 0.0 -0.5 -1.0
```

```
seq(1.32, 6.1, length.out=9) # 9 Elemente
## [1] 1.3200 1.9175 2.5150 3.1125 3.7100 4.3075 4.9050
## [8] 5.5025 6.1000
```

```
seq(1.32, 6.1, len=15) # Argumentnamen abkürzbar
```

Indexing: Submengen auswählen -> Eckige Klammern

```
vek <- c(3, 7, -2, 11, 4, 9)
```

```
vek[1] # Erstes Element zurückgeben
```

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

AltGr + 8 / 9 ,

Option + 5 / 6

```
vek[2:4] # Mehrere Elemente auswählen
```

```
## [1] 7 -2 11
```

```
vek[ c(2,5,1,6,1) ] # Flexible Reihenfolge
```

```
## [1] 7 4 3 9 3
```

```
vek[-2] # Alle Elemente außer das zweite
```

```
## [1] 3 -2 11 4 9
```

```
vek[-(1:3)] # Alle Elemente außer den ersten drei
```

```
## [1] 11 4 9
```

```
vek[-1:3] # geht nicht
```

```
## Fehler in vek[-1:3]: only 0's may be mixed with negative subscripts
```

```
-1:3 # weil -1 und 1 nicht beides erfüllt werden kann
```

```
## [1] -1 0 1 2 3
```

head/tail, str, class, length

```
a <- seq(from=1, to=100, by=0.1)
```

```
head(a) # Die ersten 6 Elemente anzeigen
```

```
## [1] 1.0 1.1 1.2 1.3 1.4 1.5
```

```
tail(a, 8) # Die letzten 8 Elemente
```

```
## [1] 99.3 99.4 99.5 99.6 99.7 99.8 99.9 100.0
```

```
a[2] <- 87 # Einzelnes Element eines Objekts ändern
```

```
head(a) # das Objekt 'a' ist jetzt anders
```

```
## [1] 1.0 87.0 1.2 1.3 1.4 1.5
```

```
str(a) # Struktur: Datentyp, [Dimension], erste Werte
```

```
## num [1:991] 1 87 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 ...
```

```
class(a) # primär: numeric, logical, factor, character
```

```
## [1] "numeric"
```

```
length(a) # Länge (Anzahl Elemente) des Vektors
```

```
## [1] 991
```

```
2 * 7  
## [1] 14
```

```
2:9 * 7      # 7 wird so oft wiederholt wie nötig  
## [1] 14 21 28 35 42 49 56 63
```

```
2:9 * c(7,1)  # Dieses Konzept heißt "Recycling"  
## [1] 14  3 28  5 42  7 56  9
```

```
2:9 * c(7,1,2) # Ergebnis mit Warnung, wenn's nicht passt  
## Warning in 2:9 * c(7, 1, 2): longer object length is  
not a multiple of shorter object length  
## [1] 14  3  8 35  6 14 56  9
```


Vektoren erstellen und indizieren:

- ▶ `c` , `:` , `rep` , `seq`
- ▶ `v[n]` , `v[-n]` , `v[m:n]` , `v[-(m:n)]`
- ▶ `head` , `tail` , `str` , `class` , `length`
- ▶ Recycling

```
Punktestand <- c(Christoph=19, Berry=17, "Anna Lena"=22)
```

Leerzeichen in Namen besser vermeiden

```
Punktestand[2] # Index: Position
```

```
## Berry  
##      17
```

```
Punktestand["Berry"] # Index: Name
```

```
## Berry  
##      17
```

```
names(Punktestand) # 'Punktestand' ist ein "named vector"
```

```
## [1] "Christoph" "Berry"      "Anna Lena"
```

```
names(Punktestand) <- LETTERS[1:3]
```

```
names(Punktestand)[2] <- "NeuerName"
```

```
Punktestand
```

```
##           A NeuerName           C  
##           19           17           22
```

0. Intro

1. Grundlagen

1.1 Syntax

1.2 Hilfe

1.3 Vektoren

1.4 Statistik

Vektor mit Körpergrößen:

```
groesse <- c(149.3, 173.6, 172.2, 172.9, 161.6, 179.2,  
             164.8, 162.8, 180.5, 165.1, 181.7, 171.4,  
             172.1, 148.1, 161.1, 171.9, 186.9) # cm
```

```
mean(groesse)    # Mittelwert
```

```
## [1] 169.1294
```

```
var(groesse)     # Varianz:  $\text{cm}^2$ 
```

```
## [1] 112.591
```

```
sd(groesse)      # Standardabweichung: cm
```

```
## [1] 10.61089
```

```
min(groesse)     # Minimum
```

```
## [1] 148.1
```

```
max(groesse)     # Maximum
```

```
## [1] 186.9
```

```
range(groesse)   # Wertebereich
```

```
## [1] 148.1 186.9
```

Statistische Maßzahlen II: auch ohne Normalverteilung

```
median(groesse) # Ausreißer-unabhängig (anders als mean)
## [1] 171.9
```

```
mad(groesse) # Median absolute deviation
## [1] 10.82298
```

```
quantile(groesse) # Anteil < bestimmte Werte
##      0%    25%    50%    75%   100%
## 148.1 162.8 171.9 173.6 186.9
```

```
quantile(groesse, probs=0.80) # 80% ist hier drunter
##      80%
## 178.08
```

```
summary(groesse)
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 148.1    162.8    171.9    169.1    173.6    186.9
```

Zeigt auch Anzahl NAs an (falls vorhanden), kann auch für Tabellen verwendet werden, siehe entsprechenden Abschnitt [3.1](#)

```
groesse <- round(groesse[1:8])      ;   groesse  
## [1] 149 174 172 173 162 179 165 163
```

```
sort(groesse) # Aufsteigend sortieren  
## [1] 149 162 163 165 172 173 174 179
```

```
sort(groesse, decreasing=TRUE)  
## [1] 179 174 173 172 165 163 162 149
```

```
order(groesse)  
## [1] 1 5 8 7 3 4 2 6
```

Das kleinste ist an Stelle 1, das zweitkleinste in `groesse[5]`, etc.

```
gewicht <- c(49, 77, 66, 91, 69, 72, 73, 74)  
gewicht[order(groesse)] # Sortieren nach Reihenfolge Größe  
## [1] 49 69 74 73 66 91 77 72
```

```
sample(0:9, size=7)                # Zufällig Werte aus Vektor ziehen  
## [1] 2 7 1 4 9 8 6
```

```
sample(0:9, size=7, replace=TRUE)  # Ziehen mit Zurücklegen  
## [1] 5 6 9 0 7 6 5
```

Kontinuierliche Verteilungen:

```
rnorm(n=5, mean=20, sd=3.5)        # aus Normalverteilung  
## [1] 21.9 19.0 20.4 19.9 11.2
```

```
rexp(n=5, rate=1/20)               # Exponentialverteilung  
## [1] 7.27 23.47 22.79 8.56 29.17
```

```
runif(n=5, min=15, max=25)         # Gleichverteilung (uniform)  
## [1] 22.8 19.3 24.3 22.7 17.6
```

```
rbeta(n=5, shape1=3, shape2=9)     # Beta-verteilung  
## [1] 0.1879 0.0687 0.2998 0.4172 0.1498
```

Diskrete Verteilungen:

```
rpois(n=5, lambda=20)              # Poisson-verteilung  
## [1] 19 13 23 29 29
```

```
rbinom(n=5, size=100, prob=1/5)    # Binomial-verteilung  
## [1] 27 16 21 27 22
```

Statistische Maßzahlen, Sortierungen und Zufallszahlen:

- ▶ `mean`, `var`, `sd`
- ▶ `min`, `max`, `range`, `median`, `quantile`, `summary`
- ▶ `round`, `sort`, `order` (decreasing)
- ▶ `sample`, `rnorm` etc


```
val <- c(1, 7, 3, 3, 8, 5, 6, 6, 6, 7)
```

```
unique(val) # ursprüngliche Reihenfolge beibehalten  
## [1] 1 7 3 8 5 6
```

```
duplicated(val)  
## [1] FALSE FALSE FALSE TRUE FALSE FALSE FALSE TRUE  
## [9] TRUE TRUE  
duplicated(val, fromLast=TRUE)  
## [1] FALSE TRUE TRUE FALSE FALSE FALSE TRUE TRUE  
## [9] FALSE FALSE
```

```
werte <- c(149.3, 173.6, 172.2, 172.9, 161.6, 179.2, 164.8, 142.8)
```

```
round(werte, digits=-1) # Auf 10er runden
```

```
## [1] 150 170 170 170 160 180 160 140
```

```
round(werte, -2) # Auf 100er runden
```

```
## [1] 100 200 200 200 200 200 200 100
```

```
round(werte/5)*5 # Auf 5er runden
```

```
## [1] 150 175 170 175 160 180 165 145
```

```
pi
## [1] 3.141593
```

Das Verhalten von R kann in vielen Optionen angepasst werden, z.B. für den Umgang mit Warnmeldungen oder Ausgaben (printed output).

`digits` regelt, wieviele relevante Nachkommastellen angezeigt werden (bezogen auf die Größenordnung der Zahl):

```
oo <- options(digits=3) # ca 2 Nachkommastellen anzeigen
oo # bisheriger Wert jetzt in oo (old options)
## $digits
## [1] 7
```

```
pi
## [1] 3.14
```

```
options(oo) ; rm(oo) # Einstellungen zurücksetzen
```

```
sample(1:50, 3)
## [1] 18 45  5
sample(1:50, 3)
## [1] 37 29 22
```

Für die Folien immer wieder die gleichen "Zufallszahlen" erzeugen
-> Startpunkt für den RNG (Random Number Generator) setzen:

```
set.seed(12345)
```

```
sample(1:50, 3)
## [1] 14 16 26
```

```
set.seed(12345)
sample(1:50, 3)
## [1] 14 16 26
```