R-Kurs: Tag 4

Lukas Mödl, Matthias Becher, Erin Sprünken

Institut für Biometrie und Klinische Epidemiologie

Charité - Universitätsmedizin Berlin, Berlin

erin-dirk.spruenken@charite.de

February 17, 2022

CHARITÉ UNIVERSITÄTSMEDIZIN BERLIN

IBKE

Installation weiterer R Pakete

Jede R Umgebung installiert und lädt standardmäßig die Pakete `base`, `stats`, `datasets`, `methods` und `graphics`.

- Installation weiterer Pakete mit:

```
install.packages("name-des-pakets", dependencies = TRUE)
```

- Bei jedem Start von R muss das Paket, wenn es verwendet werden soll, geladen werden:

```
library("name-des-pakets")
```

- Aktualisieren der Pakete mit:

```
update.packages()
```

Beispiel: Installation und Laden des R Pakets MASS

```
> install.packages("MASS")
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.1/MASS_7.3-55.zip'
Content type 'application/zip' length 1192198 bytes (1.1 MB)
downloaded 1.1 MB

package 'MASS' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\[redacted]\AppData\Local\Temp\RtmpSMaYtV\downloaded_packages
> library("MASS")
```

Empfehlenswerte Pakete

- `MatchIt` für Propensity Score Matching
- `MASS` für Negativ-binomiale Regression
- `lmer` bzw. `lme4` für Mixed-Models
- `pwr` für Power-Analyse und insbesondere zur Fallzahlplanung
- `ggplot2` für schöne Plots
- `foreign` für das Einlesen von `.sav`-Dateien
- ...

Konzeption if-Anweisung

Möchte man einen Codeblock abhängig von einem bestimmten Wert ausführen oder nicht, muss eine if-Anweisung verwendet werden. if-Anweisungen definieren einen Codeblock, der nur dann ausgeführt wird, wenn die übergebene Bedingung wahr ist.

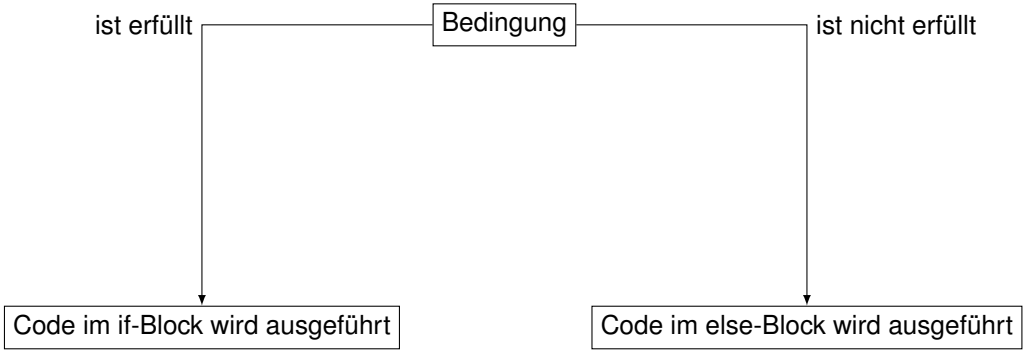
Beispiel: Bedingte Ausgabe

```
x = 2

if(x <= 3)
{
    print("x ist kleiner als 3")
}
```

Konzeption if-else-Anweisung

Bei der if-Anweisung wird der Code, der nach der if-Anweisung steht immer ausgeführt. Das ist in einer Entweder-Oder-Situation natürlich nicht wünschenswert. Hier schafft die if-else-Anweisung Abhilfe.



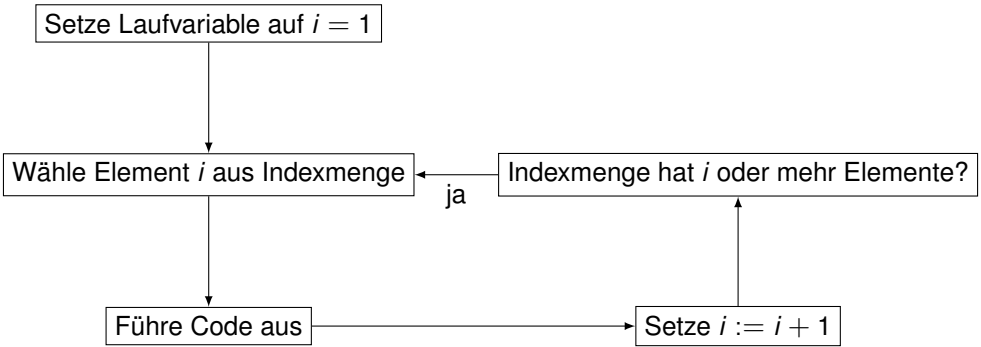
Beispiel: Bedingte Ausgabe

```
x = 2

if(x <= 3)
{
    print("x ist kleiner als 3")
}
else
{
    print("x ist größer als 3")
}
```

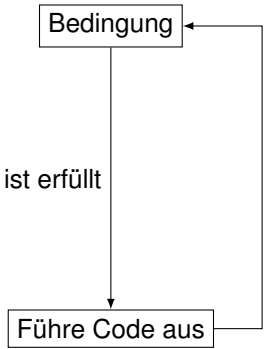
Konzeption `for`-Schleife

Eine beiden wesentlichen Schleifen in R ist die `for`-Schleife. `for`-Schleifen führen einen Codeblock so oft aus, wie eine definierte Laufvariable Werte in einer Indexmenge annehmen kann.



Konzeption `while`-Schleife

Die andere wesentliche Schleife ist die `while`-Schleife. Im Gegensatz zur `for`-Schleife wird hier ein Codeblock so lange ausgeführt, wie eine übergebene Bedingung wahr ist.



Beispiel I: Einfacher `for`-Schleife

```
indexmenge = seq(1, 10, by = 1)

for(i in indexmenge)
{

    print(i)

}
```

Beispiel II: Einfache `while`-Schleife

```
a = 10
b = 0

while(a > b)
{
    a = a - 0.5
    b = b + 0.5

    print("a = ")
    print(a)
    print("b = ")
    print(b)
}
```

Achtung vor unendlichen Schleifen!

```
a = 10
b = 0

while(a > b)
{

    a = a + 1

}
```

Diese Schleife endet nie! Zum Abbruch muss “ESC” gedrückt werden.

Beispiel III: Berechnung des Standardfehlers mittels Monte-Carlo Simulation

```
n <- 1000
stichprobe <- rnorm(n, 5, 2)
mittelwerte <- numeric(n)

for(i in seq(1, n, by = 1))
{
    s <- sample(stichprobe, size = n, replace = TRUE)
    mittelwerte[i] <- mean(s)
}

standardfehler <- sd(mittelwerte)
```

Konzeption eigener Funktionen

Bisher haben wir nur vordefinierte Funktionen wie beispielsweise `t.test` verwendet. R erlaubt es aber auch, eigene Funktionen zu schreiben.

```
meine_funktion <- function(<argumente>)
{

    <code>

    return(<funktionswert>)
}
```

Beispiel I: Eine Funktion mit einem Argument

```
f <- function(x)
{
    y <- x^2

    return(y)
}
```

Beispiel II: Eine weitere Funktion mit einem Argument

```
wochentag <- function(x)
{

    return(format(as.Date(x, "%d.%m.%Y"), "%A"))

}
```

Funktionen werden wenn möglich vektorisiert

```
daten <- c("8.5.1949", "5.3.1995", "1.1.2000", "1.1.2022")

> wochentag(daten)

[1] "Mittwoch" "Mittwoch" "Samstag" "Samstag"
```

Beispiel III: Eine Funktion mit zwei Argumenten

```
bmi_rechner <- function(gewicht, groesse) {

    # gewicht in kg
    # groesse in m

    bmi <- gewicht / groesse^2

    return(bmi)
}
```

Default Werte für Argumente vergeben

```
bmi_rechner <- function(gewicht, groesse = 1.82)
{
    # gewicht in kg
    # groesse in m

    bmi <- gewicht / groesse^2

    return(bmi)
}
```

Funktion für den BMI Rechner verbessern

```
bmi_rechner <- function(gewicht, groesse = 1.82)
{
    # gewicht in kg
    # groesse in m

    if(gewicht <= 0 | groesse <= 0)
    {
      stop("Gewicht und Größe dürfen nicht <= 0 sein!")
    }
    else
    {
      bmi <- gewicht / groesse^2
      return(bmi)
    }
```