R - Lesson 2

Dr. Katja Nieberle

2026-01-03

Lernziele

Studierende beherrschen Zuweisungen, Objektabfragen (`mode`, `str`), Workspace-Verwaltung (`ls`, `rm`) und grundlegende Datenstrukturen (Vektoren, Matrizen, Data Frames). Sie erkunden Daten mit summary, head und filtern mit Logik.

R als Taschenrechner

```
1 + 2 * 3              # 7
```

```
## [1] 7
```

```
sum(1,2,3)             # 6
```

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

```
2 * 5^2 - 10 * 5       # 230
```

```
## [1] 0
```

```
4 * sin(pi / 2)        # 4
```

```
## [1] 4
```

```
0 / 0                  # NaN (Not a Number)
```

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

```
x1 <- 3.25             # dem Objekt x1 wird 3.25 zugewiesen
x1                     # Ausgabe implizit
```

```
## [1] 3.25
```

```
print(x1)              # Ausgabe explizit
```

```
## [1] 3.25
```

R Objektzuweisungen (assignment) und Ausgabe

Verwende <- für Zuweisungen und nicht =

```
x1 <- 3.25              # dem Objekt x1 wird 3.25 zugewiesen
x2 <- 5.6 - 5

x1                      # Ausgabe implizit
```

```
## [1] 3.25
```

```
print(x1)
```

```
## [1] 3.25
```

```
print(x1+x2)            # Ausgabe explizit
```

```
## [1] 3.85
```

R Objekte, Klassen und Datentypen

```
a <- 5; mode(a)         # numeric
```

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

```
b <- "Text"; mode(b)    # character
```

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

```
c <- TRUE; mode(c)      # logical
```

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

```
avec <- c(1,2,3,4,5); mode(avec) # vector - numeric
```

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

```
print(avec)
```

```
## [1] 1 2 3 4 5
```

```
X <- matrix(1:6, nrow=2)
mode(X)                 # matrix - numeric
```

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

```r
print(X)
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```r
# Mit
# is.numeric(x),
# is.na(x)
# kann man den Datentyp überprüfen. Probier es aus!
```

NA-Behandlung

```r
x <- c(2, NA, 4)
is.na(x)               # TRUE/NA/...
```

```
## [1] FALSE  TRUE FALSE
```

```r
sum(is.na(x))          # 1
```

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

```r
mean(x, na.rm=TRUE)    # Ignoriere NA
```

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

```r
# na.omit(df)          # für data frames
```

Indizierung in Vektoren

```r
x <- c(3,6,9,8,4,1,2)  # vector
x[3]                   # 9
```

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

```r
x[c(1,3)]              # 3,9
```

```
## [1] 3 9
```

```r
x[x < 4]               # Logische Filter: 3,1,2
```

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

```r
length(x)              # 7
```

```
## [1] 7
```

```r
sum(x < 4)             # Anzahl Treffer: 3
```

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

```r
which(x < 4)           # 1,6,7
```

```
## [1] 1 6 7
```

Matrizen und Data Frames

```r
Einkaufen <- data.frame(
  Produkt = c("Apfelsaft", "Quark"),
  Menge = c(4,2)
)
str(Einkaufen)         # Struktur
```

```
## 'data.frame':    2 obs. of  2 variables:
##  $ Produkt: chr  "Apfelsaft" "Quark"
##  $ Menge  : num  4 2
```

```r
Einkaufen[, "Menge"]   # Spalte
```

```
## [1] 4 2
```

```r
Einkaufen[Einkaufen$Menge > 2, ]  # Filter
```

```
##     Produkt Menge
## 1 Apfelsaft     4
```

```r
#cbind(Einkaufen)
```

Funktionen in R: Struktur und Nutzung

R-Funktionen kapseln Code für wiederverwendbare Aufgaben. Syntax: `funktionsname(arg1 = wert1, arg2 = wert2)`.

1. **Name einer Funktion** Beispiele:

- `print()` - gibt den Wert des Argumentes aus
- `mean()` – berechnet arithmetisches Mittel. Name eindeutig, kleingeschrieben (Konvention), kann `?` oder `::` enthalten (z.B. `dplyr::filter()`, wobei `dplyr` - der Name des packages ist, aus welchem die Funktion verwendet wird).

2. **Argumente** Argumente sind Parameter mit Default-Werten. Beispiele:

- `print(x, ...)`, `x`: ein Objekt oder Methode.
- `mean(x, na.rm = FALSE, trim = 0)`, mit `x`: Numerischer Vektor (Pflicht).; `na.rm`: Logisch, NA ignorieren? (Default: FALSE); `trim`: Anteil zu kürzen (Default: 0)

```
a <- 7
print(a)
```

```
## [1] 7
```

```
print(7)
```

```
## [1] 7
```

```
mean(c(1,2,3))
```

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

```
mean(c(1,NA,3), na.rm=TRUE)
```

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

3. **Rückgabewert:** Letzter Ausdruck in der Funktion oder `return()` - oft "unsichtbar"

R Hilfe: ? und ?? in RStudio

`?funktion`: Hilfe für exakte Funktion (öffnet Help-Pane unten rechts).

`??stichwort`: Vagus-Suche über alle Pakete

Hilfe in RStudio:

- Console: ?mean tippen → Help-Tab öffnet sich.
- Rechts unten: Help-Pane (Suche oben).
- Strg+Shift+Space: Argument-Hilfe im Editor.
- F1: Hilfe zu markiertem Code.

R Packages

R Packages sind Erweiterungen mit Funktionen, Daten und kompiliertem Code. Base R hat ~30 Standardpakete, CRAN >20.000 weitere

packages sind Sammlungen von: - Funktionen (z.B. ggplot() für Diagramme) - Datensätzen (z.B. iris erweitert) - Dokumentationen (?funktion) - Compiled Code (schnellere Algorithmen)

Prozess: Install → Load → Use - Installation (einmalig) - `require()` oder `library()` - package laden pro R Session

```r
install.packages("openxlsx")
library(openxlsx)
```

```r
?mean     # Hilfe zu mean()
??durchschnitt  # Suche "durchschnitt"
help(mean)  # Äquivalent zu ?
help.search("mean")  # apropos-Suche
```

Conditionen: if-then-else

Bedingte Anweisungen prüfen Logik und führen Code aus. Eine "verkürzte" Form für Vektoren als Funktion `ifelse()`, if/else für einzelne Werte.

```r
# Einzelne Bedingung
x <- 7
if(x > 5) {
  print("x ist groß")
} else {
  print("x ist klein")
}
```

```
## [1] "x ist groß"
```

```r
# Vektorwertig (wichtig für Daten!)
score <- c(85, 92, 78, 95)
ifelse(score > 90, "Sehr gut", "Gut")
```

```
## [1] "Gut"      "Sehr gut" "Gut"      "Sehr gut"
```

```r
# Ergebnis: "Gut" "Sehr gut" "Gut" "Sehr gut"
```

Schleifen: for, while, repeat

Schleifen automatisieren Wiederholungen. `for` für bekannte Iterationenanzahl, `while` für unbekannte.

```r
# for-Schleife (über Vektor)
umsatz <- c(100, 150, 200)
for(i in umsatz) {
  print(paste("Umsatz:", i))
}
```

```
## [1] "Umsatz: 100"
## [1] "Umsatz: 150"
## [1] "Umsatz: 200"
```

```r
# while-Schleife
i <- 1
while(i <= 3) {
  print(i)
  i <- i + 1  # WICHTIG: sonst Endlosschleife!
}
```

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

```r
# repeat mit break
i <- 0
repeat {
  i <- i + 1
  if(i > 3) break
  print(i)
}
```

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

Workspace-Management - Verwalte deine Objekte

```r
ls()                    # Alle Objekte der entsprechenden directory auflisten
rm(blabla)              # Objekt löschen
getwd()                 # get working directory - Arbeitsverzeichnis
setwd("C:/data")        # set working directory - Setze das Arbeitsverzeichnis
```

Hausaufgaben:

1. Berechne `(3 + 5) * 2 - 4`. Weise das Ergebnis der Variable `result` zu. Gebe es mit `print()` und implizit aus. Erweitere um Variable `f <- 1.1` und multipliziere mit dem `result`.
2. Erstelle Vektor `sale <- c(10, NA, 15, 20, 30, 40, NA, 90)`. Prüfe `mode(sale)`, und `is.na(sale)`. Berechne `mean(sale, na.rm=TRUE)`. Was für Berechnungen hast Du genau durchgeführt?
3. Erstelle den Vektor `price <- c(2.5, 1.8, 3.2, 0.9, 4.1, 8.7, 9.1)`.

- Extrahiere 2. und 4. Element.
- Filtere `price > 2`.
- Berechne die Summe der Elemente mit `sum()`.
- Berechne die Summe der Elemente > 2.

4. Installiere `dplyr` (falls nicht vorhanden). Lade mit `library(dplyr)` und `require(dplyr)`.
5. Erstelle Funktion `calculation(sales, rate=0.19)`. Die Funktion berechnen die Umsatzsteuer als `sales * (1+rate)`. Teste mit `calculation(1000)` und mit `calculation(1000, 0.2)`.
6. Für einen sortierten Vektor `sales <- c(100,150,170,190,250)`

- schreibe eine for-Schleife, um jeden Umsatz aus `sales` zu verdoppeln und zu printen.
- schreibe eine while-Schleife, die den Umsatz solange verdoppelt und printen, solange die Werte < 200 sind.
- schreibe eine for-Schleife und entscheide innerhalb der Schleife mit einer `if-than-else` Bedingung: Die Werte > 200 sollen einzeln geprintet werden; die Werte $<= 200$ sollen doppelt geprintet werden.

Lösungen:

1.

```
result <- (3 + 5) * 2 - 4  # 14
print(result)              # 14
```

```
## [1] 12
```

```
result                     # 14 (implizit)
```

```
## [1] 12
```

```
f <- 1.1
result * f             # 15.4
```

```
## [1] 13.2
```

2.

```
sale <- c(10, NA, 15, 20, 30, 40, NA, 90)
mode(sale)                # numeric
```

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

```
is.na(sale)
```

```
## [1] FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE
```

```
mean(sale, na.rm=TRUE)
```

```
## [1] 34.16667
```

3.

```
price <- c(2.5, 1.8, 3.2, 0.9, 4.1, 8.7, 9.1)
price[c(2,4)]
```

```
## [1] 1.8 0.9
```

```
price[price > 2]
```

```
## [1] 2.5 3.2 4.1 8.7 9.1
```

```r
sum(price > 2)
```

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

4.

```r
install.packages("dplyr")  # Falls nötig
library(dplyr)             # Kein Return
require(dplyr)             # TRUE
```

5.

```r
calculation <- function(sales, rate=0.19) {
  result <- sales * (1 + rate)
  return(result)
}
calculation(1000)
```

```
## [1] 1190
```

```r
calculation(1000, 0.2)
```

```
## [1] 1200
```

6.

```r
sales <- c(100,150,170,190,250,300)
for(i in sales) {
  print(i * 2)
}
```

```
## [1] 200
## [1] 300
## [1] 340
## [1] 380
## [1] 500
## [1] 600
```

```r
i <- 1   # Startwert (erster sales-Wert)
while(sales[i] < 200) {
  print(sales[i] * 2)
  i = i + 1
}
```

```
## [1] 200
## [1] 300
## [1] 340
## [1] 380
```

```
for(i in sales) {
  if (i > 200)
    print(i)
  else print (i*2)
}
```

```
## [1] 200
## [1] 300
## [1] 340
## [1] 380
## [1] 250
## [1] 300
```