
# Programmierung und Deskriptive Statistik

BSc Psychologie WiSe 2025/26

Belinda Fleischmann und Dirk Ostwald

| | Gruppe 1/2 | Gruppe 3 | Format | Thema |
|---|---|---|---|---|
| 1 | Mi, 15.10. | Do, 16.10. | Seminar | (1) Grundbegriffe der Informatik |
| 2 | Mi, 22.10. | Do, 23.10. | Seminar | (2) Arithmetik und Variablen |
| 3 | Mi, 29.10. | Do, 30.10. | Übung | (2) Arithmetik und Variablen |
| **4** | **Mi, 05.11.** | **Do, 06.11.** | **Seminar** | **(3) Vektoren und Matrizen** |
| 5 | Mi, 12.11. | Do, 13.11. | Übung | (3) Vektoren und Matrizen |
| 6 | Mi, 19.11. | Do, 20.11. | Seminar | (4) Listen und Dataframes |
| 7 | Mi, 26.11. | Do, 27.11. | Übung | (4) Listen und Dataframes |
| 8 | Mi, 03.12. | Do, 04.12. | Seminar | (5) Datenmanagement |
| 9 | Mi, 10.12. | Do, 11.12. | Übung | (5) Datenmanagement |
| | Mo, 15.12 | | Abgabe | *Individualleistung (1/2)* |
| 10 | Mi, 17.12. | Do, 18.12. | Seminar | (6) Strukturiertes Progr.: Kontrollfluss, Debugging |
| 11 | Mi, 07.01. | Do, 08.01. | Seminar | (7) Häufigkeitsverteilungen |
| 12 | Mi, 14.01. | Do, 15.01. | Übung | (7) Häufigkeitsverteilungen |
| 13 | Mi, 21.01. | Do, 22.01. | Seminar | (8) Maße der zentralen Tendenz und Datenvariabilität |
| 14 | Mi, 28.01. | Do, 29.01. | Übung | (8) Maße der zentralen Tendenz und Datenvariabilität |
| | Mo, 02.02 | | Abgabe | *Individualleistung (2/2)* |

# (3) Vektoren und Matrizen

## Vektoren

- Übersicht und Erzeugung
- Charakterisierung
- Indizierung
- Arithmetik
- Attribute

## Matrizen

- Übersicht und Erzeugung
- Charakterisierung
- Indizierung
- Arithmetik
- Attribute

**Vektoren**

- Übersicht und Erzeugung
- Charakterisierung
- Indizierung
- Arithmetik
- Attribute

Matrizen

- Übersicht und Erzeugung
- Charakterisierung
- Indizierung
- Arithmetik
- Attribute

## Vektoren

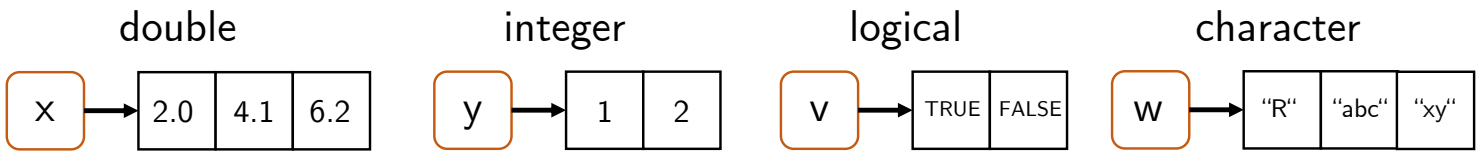
- **Übersicht und Erzeugung**
- Charakterisierung
- Indizierung
- Arithmetik
- Attribute

## Matrizen

- Übersicht und Erzeugung
- Charakterisierung
- Indizierung
- Arithmetik
- Attribute

# Übersicht

- R operiert mit *Datenstrukturen* (z.B. Vektoren, Matrizen, Listen und Dataframes).
- Die einfachste dieser Datenstrukturen ist ein Vektor.
- Vektoren sind geordnete Folgen von Datenwerten, die in einem Objekt zusammengefasst sind und einem Variablennamen zugewiesen sind.
- Die einzelnen Datenwerte eines Vektors heißen *Elemente* des Vektors.
- Vektoren, deren Elemente alle vom gleichen Datentyp sind, heißen **atomar**.
- Die zentralen Datentypen sind **numeric** (**double**, **integer**), **logical**, **character**




- Mit dem Begriff **Vektor** ist hier immer ein **atomarer Vektor** gemeint.

# Erzeugung einelementiger Vektoren (Elementarwerte)

## **Numeric** (double, integer)

Per default werden numerische Werte (mit oder ohne Dezimalstellen) als double initialisiert. Dezimalzahlen können in Dezimalnotation oder wissenschaftlicher Notation spezifiziert werden. Weitere mögliche Werte sind Inf, -Inf, und NaN (Not-a-Number).

```
h <- 1                  # Einelementiger Vektor vom Typ double (1)
i <- 2.1e2              # Einelementiger Vektor vom Typ double (210)
j <- 2.1e-2             # Einelementiger Vektor vom Typ double (0.021)
k <- Inf                # Einelementiger Vektor vom Typ double (unendlich)
l <- NaN                # Einelementiger Vektor vom Typ double (NaN)
```

Integer werden wie double ohne Dezimalstellen spezifiziert, gefolgt von einem L (long integer).

```
x <- 1L                 # Einelementiger Vektor vom Typ integer
y <- 200L               # Einelementiger Vektor vom Typ integer
```

## Logical

TRUE oder FALSE, abgekürzt T oder F.

```
x <- TRUE               # Einelementiger Vektor vom Typ logical
y <- F                  # Einelementiger Vektor vom Typ logical
```

## Character

Anführungszeichen (“a”) oder Hochkommata (‘a’).

```
x <- "a"                # Einelementiger Vektor vom Typ character
y <- 'test'             # Einelementiger Vektor vom Typ character
```

# Erzeugung mehrelementiger Vektoren

## Direkte Konkatenation von Elementarwerten mit `c()`

```r
x <- c(1, 2, 3)                 # numeric vector [1,2,3]
y <- c(0, x, 4)                 # numeric vector [0,1,2,3,4]
s <- c("a", "b", "c")           # character vector ["a", "b", "c"]
l <- c(TRUE, FALSE)             # logical vector [TRUE, FALSE]
```

- Beachte: `c()` konkateniert die Eingabeargumente und erzwingt einen einheitlichen Datentyp (vgl. coercion)

```r
x <- c(1, "a", TRUE)            # character vector ["1", "a", "TRUE"]
```

## Erzeugen "leerer" Vektoren mit `vector()`

```r
v <- vector("double", 3)        # double vector [0,0,0]
w <- vector("integer", 3)       # integer vector [0,0,0]
l <- vector("logical", 2)       # logical vector [FALSE, FALSE]
s <- vector("character", 4)     # character vector ["", "", "", ""]
```

## Erzeugen "leerer" Vektoren mit double(), integer(), logical(), character()

```r
v <- double(3)                  # double vector [0,0,0]
w <- integer(3)                 # integer vector [0,0,0]
l <- logical(2)                 # logical vector [FALSE, FALSE]
s <- character(4)               # character vector ["", "", "", ""]
```

# Erzeugung von Vektoren als Sequenzen

Erzeugen von ganzzahligen Sequenzen mithilfe des **Colonoperators** :

`a:b` erzeugt ganzzahlige Sequenzen von a (inklusive) bis b (maximal)

```r
x <- 0:5                    # [0, 1, 2, 3, 4, 5]
y <- 1.5:6.1                # [1.5, 2.5, 3.5, 4.5, 5.5]
```

Erzeugen von Sequenzen mit `seq()`

`seq(from, to, by = ((to - from)/(len - 1), len = NULL, ...)`

```r
x_1 <- seq(0, 5)            # wie `0:5`, ganzzahlige Sequenz zw. 0 (inkl.) und 5 (max)
                            # [0, 1, 2, 3, 4, 5]
x_2 <- seq(0, 1, len = 5)   # 5 Zahlen zwischen 0 (inkl.) und 1 (inkl.), equidistant
                            # [0.00, 0.25, 0.50, 0.75, 1.00]
x_3 <- seq(0, 2, by = .15)  # 0.15 Schritte zwischen 0 (inkl.) und 2 (max.)
                            # [0.00, 0.15, 0.30, ..., 1.50 1.65 1.80 1.95]
x_4 <- seq(1, 0, by = -.1)  # -0.1 Schritte zwischen 1 (inkl.) und 0 (min.)
                            # [1.0 0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1 0.0]
```

`seq.int()`, `seq_len()`, `seq_along()` als weitere Varianten

```r
x_1 <- seq.int(0, 5)        # wie `0:5`, ganzzahlige Sequenz zw. 0 (inkl.) und 5 (max)
                            # [0, 1, 2, 3, 4, 5]
x_2 <- seq_len(5)           # Natürliche Zahlen bis 5, [1, 2, 3, 4, 5]
x_3 <- seq_along(c("a", "b"))  # wie `seq_len(length(c("a", "b")))`
```

## Vektoren

- Übersicht und Erzeugung
- **Charakterisierung**
- Indizierung
- Arithmetik
- Attribute

## Matrizen

- Übersicht und Erzeugung
- Charakterisierung
- Indizierung
- Arithmetik
- Attribute

# Vektoreigenschaften ausgeben

## `length()` gibt die Anzahl der Elemente eines Vektors aus

```
x <- 0:10          # Vektor
length(x)          # Anzahl der Elemente des Vektors
```

```
[1] 11
```

## `typeof()` gibt den elementaren Datentyp eines Vektors aus

```
x <- 1:3L          # Vektor
typeof(x)          # Datentyp des atomic vectors
```

```
[1] "integer"
```

```
y <- c(T, F, T)    # Vektor
typeof(y)          # Datentyp des atomic vectors
```

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

Anmerkung: `mode()` und `storage.mode()` werden nicht empfohlen, sie existieren für S Kompatibilität.

## `is.logical()`, `is.double()`, `is.integer()`, `is.character()` testen den Datentyp

```
is.double(x)       # Testen, ob x vom Typ double ist
```

```
[1] FALSE
```

```
is.logical(y)      # Testen, ob y vom Typ logical ist
```

```
[1] TRUE
```

# Datentypangleichung (Coercion)

Bei Konkatenation verschiedener Datentypen wird ein einheitlicher Datentyp erzwungen. Es gilt

$$\text{character} > \text{double} > \text{integer} > \text{logical}$$

```
x <- c(1.2, "a")      # Kombination gemischter Datentypen (character schlägt double)
x
```

```
[1] "1.2" "a"
```

```
typeof(x)             # Erzeugter Vektor ist vom Datentyp character
```

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

```
y <- c(1L, TRUE)      # Kombination  gemischter Datentypen (integer schlägt logical)
y
```

```
[1] 1 1
```

```
typeof(y)             # Erzeugter Vektor ist vom Typ integer
```

```
[1] "integer"
```

# Datentypangleichung (Coercion)

**Explizite** Coercion mit `as.logical()`, `as.integer()`, `as.double()`, `as.character()`

```
x <- c(0, 1, 1, 0)          # double Vektor
y <- as.logical(x)          # Umwandlung in logical Vektor
y
```

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

Coercion geschieht aber auch oft **implizit**:

```
x <- c(T, F, T, T)          # logical Vektor
s <- sum(x)                 # Summation wandelt logical Elemente automatisch in integer
s
```

```
[1] 3
```

## Vektoren

- Übersicht und Erzeugung
- Charakterisierung
- **Indizierung**
- Arithmetik
- Attribute

## Matrizen

- Übersicht und Erzeugung
- Charakterisierung
- Indizierung
- Arithmetik
- Attribute

# Indizierung

## Grundlagen

- Einzelne oder mehrere Vektorkomponenten werden durch Indizierung adressiert.
- Indizierung wird auch Indexing, Subsetting, oder Slicing genannt.
- Zur Indizierung werden eckige Klammern [ ] benutzt.
- Indizierung kann zur Kopie oder Manipulation von Komponenten benutzt werden.
- Der Index des ersten Elements ist 1 (nicht 0, wie in anderen Sprachen).

## Beispiel

```
x <- c("a", "b", "c")   # character vector ["a", "b", "c"]
y <- x[2]               # Kopie von "b" (neues Object), referenziert von y
x[3] <- "d"             # Änderung von x zu x = ["a", "b", "d"]
```

## Prinzipien der Indizierung in R

- Ein **Vektor positiver Zahlen** adressiert entsprechende Komponenten.
- Ein **Vektor negativer Zahlen** adressiert komplementäre Komponenten.
- Ein **logischen Vektor** adressiert die Komponenten mit TRUE.
- Ein **character Vektor** adressiert benannte Komponenten.

# Beispiele

## Indizierung mit einem **Vektor positiver Zahlen**

```r
x <- c(1, 4, 9, 16, 25)   # [1, 4, 9, 16, 25]
y <- x[1:3]               # `1:3` erzeugt Vektor [1, 2, 3]; x[1:3] = x[c(1, 2, 3)] = [1, 4, 9]
z <- x[c(1, 3, 5)]        # `c(1, 3, 5)` erzeugt Vektor [1, 3, 5]; x[c(1, 3, 5)] = [1, 9, 25]
```

## Indizierung mit einem **Vektor negativer Zahlen**

```r
x <- c(1, 4, 9, 16, 25)   # [1, 4, 9, 16, 25]
y <- x[c(-2, -4)]         # Alle Komponenten außer 2 und 4, x[c(-2, -4)] = [1, 9, 25]
z <- x[c(-1, 2)]          # Gemischte Indizierung nicht erlaubt (Fehlermeldung)
```

## Indizierung mit einem **logischen Vektor**

```r
x <- c(1, 4, 9, 16, 25)   # [1, 4, 9, 16, 25]
y <- x[c(T, T, F, F, T)]  # Nur die TRUE Komponenten,  x[c(T, T, F, F, T)] = [1, 4, 25]
z <- x[x > 5]             # x > 5 = [F, F, T, T, T], x[x > 5] = [9, 16, 25]
```

## Indizierung mit einem **character Vektor**

```r
x <- c(1, 4, 9, 16, 25)   # [1, 4, 9, 16, 25]
names(x) <- c("a","b")    # Benennung der Komponenten als [a  b <NA> <NA> <NA>]
y <- x["a"]               # x["a"] = 1
```

# Anmerkungen zur Indizierung in R

## R hat eine (zu) hohe Flexibilität bei Indizierung

Out-of-range Indizes verursachen keine Fehler, sondern geben `NA` aus

```
x <- c(1, 4, 9, 16, 25)  # [1, 4, 9, 16, 25] = [1^2, 2^2, 3^2, 4^2, 5^2]
y <- x[10]               # x[10] = NA (Not Applicable)
```

Nichtganzzahlige Indizes verursachen keine Fehler, sondern werden abgerundet

```
y <- x[4.9]              # x[4.9] = x[4] = 16
z <- x[-4.9]             # x[-4.9] = x[-4] = [1, 4, 9, 25]
```

Leere Indizes indizieren den gesamten Vektor

```
y <- x[]                 # y = x
```

**Vektoren**

- Übersicht und Erzeugung
- Charakterisierung
- Indizierung
- **Arithmetik**
- Attribute

Matrizen

- Übersicht und Erzeugung
- Charakterisierung
- Indizierung
- Arithmetik
- Attribute

# Elementweise Auswertung

**Unitäre** arithmetische Operatoren und Funktionen werden elementweise ausgewertet

```
a <- seq(0, 1, len = 11) # a = [ 0.0, 0.1 , ..., 0.9, 1.0]
b <- -a                  # b = [-0.0, -0.1, ..., -0.9, -1.0]
v <- a^2                 # v = [ 0.0^2 ,  0.1^2 , ..., 0.9^2, 1.0^2]
w <- log(a)              # w = [ln(0.0), ln(0.1), ..., ln(0.9), ln(1.0)]
```

**Binäre** arithmetische Operatoren werden elementweise ausgewertet

Vektoren gleicher Länge

```
a <- c(1, 2, 3)          # a = [1,2,3]
b <- c(2, 1, 4)          # b = [2,1,4]
v <- a + b               # v = [1,2,3] + [2,1,4] = [1+2,2+1,3+4] = [3,3,7]
w <- a - b               # w = [1,2,3] - [2,1,4] = [1-2,2-1,3-4] = [-1, 1, -1]
x <- a * b               # x = [1,2,3] * [2,1,4] = [1*2,2*1,3*4] = [2, 2, 12]
y <- a / b               # y = [1,2,3] / [2,1,4] = [1/2,2/1,3/4] = [0.50, 2, 0.75]
```

Vektoren und Skalare

```
a <- c(1, 2, 3)          # a = [1,2,3]
b <- 2                   # b = [2]
v <- a + b               # v = [1,2,3] + [2,2,2] = [1+2,2+2,3+2] = [3, 4, 5]
w <- a - b               # w = [1,2,3] - [2,2,2] = [1-2,2-2,3-2] = [-1, 2, 1]
x <- a * b               # x = [1,2,3] * [2,2,2] = [1*2,2*2,3*2] = [2, 4, 6]
y <- a / b               # y = [1,2,3] / [2,2,2] = [1/2,2/2,3/2] = [0.5, 1, 1.5]
```

# Recycling

- R erlaubt (leider) auch Arithmetik mit Vektoren unterschiedlicher Länge
- Bei ganzzahligen Vielfachen der Länge wird der kürzere Vektor wiederholt.

```
x <- 1:2                # x = [1, 2], length(x) = 2
y <- 3:6                # y = [3, 4, 5, 6], length(y) = 4
v <- x + y              # v = [1, 2, 1, 2] + [3, 4, 5, 6] = [4, 6, 6, 8]
```

- Arithmetik von Vektoren und Skalaren ist ein Spezialfall dieses Prinzips.

```
x <- 1:3                # x = [1, 2, 3], length(x) = 3
y <- 2                  # y = 2, length(y) = 1. y ist ein Skalar.
v <- x + y              # v = [1, 2, 3] + [2, 2, 2] = [3, 4, 5]
```

- Bei nicht ganzzahligen Vielfachen der Länge werden die ersten Komponenten des kürzeren Vektors wiederholt.

```
x <- c(1, 3, 5)         # x = [1, 3, 5], length(x) = 3
y <- c(2, 4, 6, 8, 10)  # y = [2, 4, 6, 8, 10], length(y) = 5
v <- x + y              # v = [1, 3, 5, 1, 3] + [2, 4, 6, 8, 10] = [3, 7, 11, 9, 13]
```

**Generell sollten nur Vektoren gleicher Länge arithmetisch verknüpft werden!**

# Fehlende Werte (NA)

- NA steht für "**N**ot **A**vailable" und repräsentiert fehlende Werte in R
- Das Rechnen mit NAs ergibt (meist) wieder NA.

```
3 * NA                  # Multiplikation eines NA Wertes ergibt NA
```

```
[1] NA
```

```
NA < 2                  # Relationaler Vergleich eines NA Wertes ergibt NA
```

```
[1] NA
```

```
NA^0                    # NA hoch 0 ergibt 1, weil jeder Wert hoch 0 eins ergibt (?)
```

```
[1] 1
```

```
NA & FALSE              # NA UND FALSE ergibt FALSE
```

```
[1] FALSE
```

# Fehlende Werte (NA)

- Auf NA testet man mit is.na() oder anyNA()

```
x <- c(NA, 5, NA, 10)  # Vektor mit NAs
print(x)
```

```
[1] NA  5 NA 10
```

```
x == NA                # Relationaler Vergleich mit NA ergibt NA
```

```
[1] NA NA NA NA
```

```
is.na(x)               # Logisches Testen auf NA für jedes Element
```

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

```
anyNA(x)               # Logisches Testen auf mind. 1 NA in gesamtem Objekt
```

```
[1] TRUE
```

# Ungültige Zahlenwerte (`NaN`)

- `NaN` steht für "**N**ot **a** "**N**umber" repräsentiert ungültige Zahlenwerte, z.B. durch undefinierte arithmetische Operationen

```
0 / 0                # 0 durch 0 teilen ist nicht definiert
```

```
[1] NaN
```

```
sqrt(-1)             # Wurzel einer negativen reellen Zahl ist nicht definiert
```

```
[1] NaN
```

- In Mathematik und Informatik ist Dividieren durch Null undefiniert. R verwendet jedoch das Konzept "Unendlichkeit": positive Zahlen gehen bei Division durch Null gegen positive Unendlichkeit `Inf`, negative gegen negative `-Inf`, ähnlich wie Grenzwerte in der Analysis.

```
4 / 0                # Positive reelle Zahlen durch 0 teilen ist nicht definiert
```

```
[1] Inf
```

```
-3 / 0               # Negative reelle Zahlen durch 0 teilen ist nicht definiert
```

```
[1] -Inf
```

# Ungültige Zahlenwerte (NaN)

- Das Rechnen mit NaN ergibt (meist) wieder NaN.

```
3 * NaN                 # Multiplikation eines NaN Wertes ergibt NaN
```

```
[1] NaN
```

```
NaN^0                   # NaN hoch 0 ergibt 1, weil jeder Wert hoch 0 eins ergibt (?)
```

```
[1] 1
```

```
NaN < 2                 # Relationaler Vergleich eines NaN Wertes ergibt NA
```

```
[1] NA
```

```
NaN & FALSE             # NaN UND FALSE ergibt FALSE
```

```
[1] FALSE
```

```
NaN & TRUE              # NaN UND TRUE ergibt NA
```

```
[1] NA
```

# Ungültige Zahlenwerte (NaN)

- Auf NaN testet man mit `is.nan()`

```
y <- c(NaN, 4, NaN, 3)   # Vektor mit NaNs
print(y)
```

```
[1] NaN   4 NaN   3
```

```
y == NaN                 # Relationaler Vergleich mit NaN ergibt NAs
```

```
[1] NA NA NA NA
```

```
is.nan(y)                # Logisches Testen auf NaN für jedes Element
```

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

```
any(is.nan((y)))         # Logisches Testen auf mind. 1 NaN in gesamtem Objekt
```

```
[1] TRUE
```

# Leere Werte (NULL)

- NULL zeigt die völlige Abwesenheit eines Werts an, oft genutzt für leere Objekte.

```
w <- c()                # Leerer Vektor
print(w)
```

```
NULL
```

```
z <- c(1, 2, NULL)      # NULL als Vektorelement wird ignoriert
print(z)                # Vektor hat nur die Elemete (1, 2)
```

```
[1] 1 2
```

```
print(length(z))        # Die Länge des Vektors beträgt 2
```

```
[1] 2
```

# Leere Werte (NULL)

- Das Rechnen mit NULL ergibt leere Objekte

```
3 * NULL                    # Multiplikation mit NULL ergibt einen leeren numerischen Vektor
```

```
numeric(0)
```

```
NULL < 2                    # Relationaler Vergleich mit NULL ergibt leeren logischen Vektor
```

```
logical(0)
```

- Auf NULL testet man mit `is.null()`

```
is.null(NULL)               # Logisches Testen auf NULL
```

```
[1] TRUE
```

```
is.null(logical(0))         # Leere Vektoren sind nicht äquivalent zu NULL
```

```
[1] FALSE
```

# Unterschiede zwischen `NA`, `NaN` und `NULL`

- `NA` repräsentiert allgemein **fehlende** Werte.
- `NaN` repräsentiert spezifisch **ungültige** numerische Werte, etwa als Ergebnis undefinierter arithmetischer Operationen.
- `NULL` repräsentiert ein **leeres Objekt**.
- `NaN` ist ein Sonderfall von `NA`, während `NULL` strukturell „nichts“ repräsentiert.
- Alle drei Begriffe `NA`, `NaN` und `NULL` zählen zu den *reserved words* (siehe `?reserved`).
- `NaN` wird auch von `is.na()` als fehlender Wert erfasst. Umgekehrt zählt `is.nan()` jedoch nur numerisch ungültige Werte (`NaN`) und schließt allgemeine fehlende Werte (`NA`) aus.

```r
y <- c(NaN, 4, NaN, 3)     # Vektor mit NaNs
is.na(y)                   # Logisches Testen auf NA für jedes Element
```

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

```r
x <- c(NA, 5, NA, 10)      # Vektor mit NAs
is.nan(x)                  # Logisches Testen auf NaN für jedes Element
```

```
[1] FALSE FALSE FALSE FALSE
```

## **Vektoren**

- Übersicht und Erzeugung
- Charakterisierung
- Indizierung
- Arithmetik
- **Attribute**

## Matrizen

- Übersicht und Erzeugung
- Charakterisierung
- Indizierung
- Arithmetik
- Attribute

# Attribute

## Attribute sind Metadaten von R Objekten in Form von Schlüssel-Wert-Paaren

### Attribute ausgeben lassen mit `attributes()`

```
a <- 1:3                # Ein numerischer Vektor
attributes(a)           # Aufrufen aller Attribute
```

```
NULL
```

→ Atomic vectors haben per se keine Attribute

### Attribute aufrufen und definieren mit `attr()`

```
attr(a, "S") <- "W"     # a bekommt Attribut mit Schluessel S und Wert W
attr(a, "S")            # Das Attribut mit Schluessel S hat jetzt den Wert W
```

```
[1] "W"
```

```
attributes(a)
```

```
$S
[1] "W"
```

#### Anmerkung

Attribute werden bei Operationen oft entfernt (Ausnahmen sind `names` und `dim`)

```
b <- a[1]               # Kopie des ersten Elements von a in Vektor b
attributes(b)           # Aufrufen aller Attribute von b
```

```
NULL
```

# Vektor-Elemente bezeichnen

## Spezifikation des Attributs `names` gibt den Elementen eines Vektors Namen

```
v <- c(x = 1, y = 2, z = 3)  # Elementnamengenerierung bei Vektorerzeugung
print(v)                     # Vektorausgabe
```

```
x y z
1 2 3
```

## Die Namen können zur Indizierung benutzt werden

```
v["y"]                       # Indizierung per Namen
```

```
y
2
```

## Zum Definieren und zum Aufrufen von Namen kann auch `names()` benutzt werden

```
y <- 4:6                     # Erzeugung eines Vektors
names(y) <- c("a", "b", "c") # Definition von Namen
names(y)                     # Elementnamenaufruf
```

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

## Benannte Namen können hilfreich sein, wenn der Vektor eine Sinneinheit bildet

```
p <- c(age    = 31,          # Alter (Jahre), Groesse (cm), Gewicht (kg) einer Person
       height = 198,
       weight = 75)
print(p)                     # Vektorausgabe
```

```
   age height weight
    31    198     75
```

# Exkurs: Styleguide für Zeilenumbrüche in R Code

Es kommt häufig vor, dass der vollständige Befehl nicht in eine Zeile passt. Um Fehler zu vermeiden und die Lesbarkeit zu gewährleisten, gibt es gängige Formatierleitlinien, wenn Code auf mehrere Zeilen verteilt werden muss:

Funktionsaufrufe. Argumente über mehrere Zeilen verteilen und einrücken

```
print(
  pi,                  # Das Objekt, das ausgegeben werden soll; Komma da 2. Arg folgt!
  digits=3             # Die Anzahl an digits
)
```

```
[1] 3.14
```

Verschachtelte Funktionen. Funktionen und Argumente über mehrere Zeilen verteilen und einrücken

```
round(                 # Äußere Funktion (Ebene 0)
  sum(                 # Innere Funktion (Ebene 1)
    c(7.421, 9.234)    # Vektor als Argument der inneren Funktion (Ebene 2)
  ),                   # Ende der inneren Funktion (Ebene 1)
  digits = 1           # Zweites Argument der äußeren Funktion
)                      # Ende der äußeren Funktion (Ebene 0)
```

```
[1] 16.7
```

- Das Ergebnis der inneren Funktion `sum()` ist das erste Argument der äußeren Funktion `round()`.
- Zeile 3 (`c(7.421, 9.234)`) ist genau genommen ebenfalls eine Funktion (Vektorbildung), deren Ergebnis das Argument der inneren Funktion `sum()` ist.

Vektoren

- Übersicht und Erzeugung
- Charakterisierung
- Indizierung
- Arithmetik
- Attribute

**Matrizen**

- Übersicht und Erzeugung
- Charakterisierung
- Indizierung
- Arithmetik
- Attribute

Vektoren

- Übersicht und Erzeugung
- Charakterisierung
- Indizierung
- Arithmetik
- Attribute

## Matrizen

- **Übersicht und Erzeugung**
- Charakterisierung
- Indizierung
- Arithmetik
- Attribute

# Übersicht

Matrizen sind zweidimensionale, rechteckige Datenstrukturen der Form

$$M = \begin{pmatrix} m_{11} & m_{12} & \cdots & m_{1n_c} \\ m_{21} & m_{22} & \cdots & m_{2n_c} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n_r1} & m_{n_r2} & \cdots & m_{n_rn_c} \end{pmatrix} \tag{1}$$

- Die Elemente $m_{ij}, i = 1, ..., n_r, j = 1, ..., n_c$ sind vom gleichen Typ.
- $n_r$ ist die Anzahl der Zeilen (rows), $n_c$ ist die Anzahl der Spalten (columns).
- Jedes Element einer Matrix hat einen Zeilenindex $i$ und einen Spaltenindex $j$.
- Intuitiv sind Matrizen numerisch indizierte Tabellen.
- Formal sind Matrizen in R zweidimensional interpretierte atomare Vektoren.
- Matrizen in R sind nicht identisch mit dem mathematischen Matrixbegriff.
- Matrizen in R können allerdings für Lineare Algebra verwendet werden.
- Lineare Algebra ist die Sprache (linearer) statistischer Modelle.

# Erzeugung mit `matrix()`

## Die `matrix()` Funktion befüllt Matrizen mit Vektorelementen

`matrix(data, nrow, ncol, byrow)`

```
matrix(c(1:12), nrow = 3)                # 3 x 4 Matrix der Zahlen 1,...,12, byrow = FALSE
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

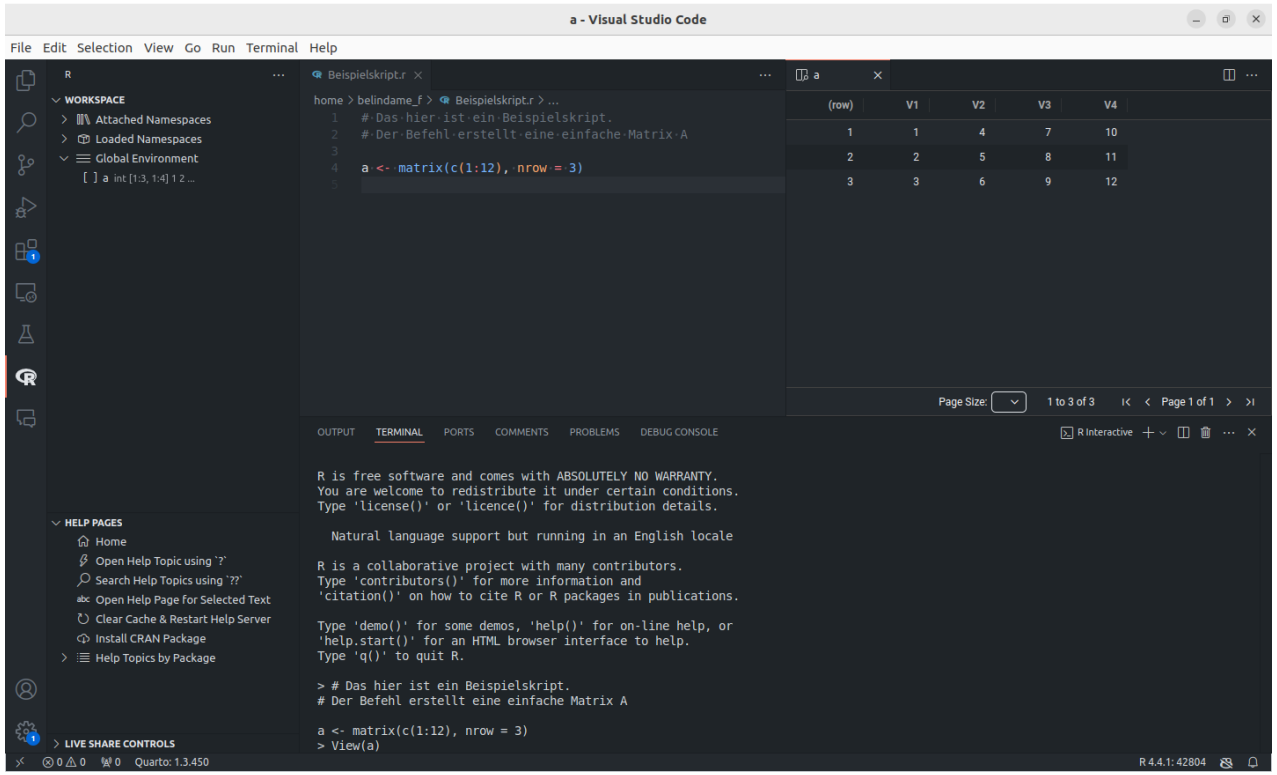
```
matrix(c(1:12), ncol = 4)                # 3 x 4 Matrix der Zahlen 1,...,12, byrow = FALSE
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

```
matrix(c(1:12), nrow = 3, byrow = TRUE) # 3 x 4 Matrix der Zahlen 1,...,12, byrow = TRUE
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

## Table Viewer




Mit dem Befehl `View()` oder im R WORKSPACE → Global Environment über das View Symbol  neben entsprechendem Objekt öffnen.

VS Code Wiki - Interactive viewers

# Erzeugung mit `cbind()`

Die Funktion `cbind()` konkateniert passende Matrizen spaltenweise (*column-bind*)

```
A <- matrix(c(1:4) , nrow = 2)        # 2 x 2 Matrix der Zahlen 1,...,4
print(A)
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

```
B <- matrix(c(5:10), nrow = 2)        # 2 x 3 Matrix der Zahlen 5,...,10
print(B)
```

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

```
C <- cbind(A, B)                      # Spaltenweise Konkatenierung von A und B
print(C)
```

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

# Erzeugung mit `rbind()`

Die Funktion `rbind()` konkateniert passende Matrizen reihenweise (*row-bind*)

```
A <- matrix(c(1:6) , nrow = 2, byrow = T)  # 2 x 3 Matrix der Zahlen 1,...,6
print(A)
```

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

```
B <- matrix(c(7:9), nrow = 1)              # 1 x 3 Matrix der Zahlen 5,...,10
print(B)
```

```
     [,1] [,2] [,3]
[1,]    7    8    9
```

```
C <- rbind(A, B)                           # reihenweise Konkatenierung von A und B
print(C)
```

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

Vektoren

- Übersicht und Erzeugung
- Charakterisierung
- Indizierung
- Arithmetik
- Attribute

## Matrizen

- Übersicht und Erzeugung
- **Charakterisierung**
- Indizierung
- Arithmetik
- Attribute

# Charakterisierung

`typeof()` gibt den elementaren Datentyp einer Matrix aus

```
A <- matrix(c(T, T, F, F), nrow = 2)        # 2 x 2 Matrix von Elementen vom Typ logical
typeof(A)
```

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

```
B <- matrix(c("a", "b", "c"), nrow = 1)     # 1 x 3 Matrix von Elementen vom Typ character
typeof(B)
```

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

`nrow()` und `ncol()` geben die Zeilen- bzw. Spaltenanzahl aus

```
C <- matrix(1:12, nrow = 3)                 # 3 x 4 Matrix
nrow(C)                                     # Anzahl Zeilen
```

```
[1] 3
```

```
ncol(C)                                     # Anzahl Spalten
```

```
[1] 4
```

Vektoren

- Übersicht und Erzeugung
- Charakterisierung
- Indizierung
- Arithmetik
- Attribute

**Matrizen**

- Übersicht und Erzeugung
- Charakterisierung
- **Indizierung**
- Arithmetik
- Attribute

# Indizierung

## Generell gilt

- Matrixelemente werden mit einen Zeilenindex und einem Spaltenindex indiziert.
- Die Indexreihenfolge ist immer 1. Zeile, 2. Spalte.
- Die Prinzipien der Indizierung entsprechen der Vektorindizierung.
- Indizes verschiedener Dimensionen können unterschiedlich indiziert werden.
- Eindimensionale Resultate liegen als Vektor, nicht als Matrix vor.

### Beispiele

```
A <- matrix(c(2:7)^2, nrow = 2)      # 2 x 3 Matrix der Zahlen 2^2,...,7^2
print(A)
```

```
     [,1] [,2] [,3]
[1,]    4   16   36
[2,]    9   25   49
```

```
a_13 <- A[1, 3]                      # Element in 1. Zeile, 3. Spalte von A [36]
a_22 <- A[2, 2]                      # Element in 2. Zeile, 2. Spalte von A [25]
a_2. <- A[2,]                        # Alle Elemente der 2. Zeile [9,25,49]
a_.3 <- A[,3]                        # Alle Elemente der 3. Spalte [36,49]
A_12 <- A[1:2, 1:2]                  # Submatrix der ersten zwei Zeilen und Spalten
A10  <- A[A>10]                      # Elemente von A groesser 10 [16,25,36,49]
A_13 <- A[1, c(F, F, T)]             # Element in 1. Zeile, 3. Spalte von A [36]
```

Vektoren

- Übersicht und Erzeugung
- Charakterisierung
- Indizierung
- Arithmetik
- Attribute

**Matrizen**

- Übersicht und Erzeugung
- Charakterisierung
- Indizierung
- **Arithmetik**
- Attribute

# Unitäre arithmetische Operationen

**Unitäre** arithmetische Operatoren und Funktionen werden elementweise ausgewertet.

```
A <- matrix(c(1:4), nrow = 2)  # 2 x 2 Matrix der Zahlen 1,2,3,4
print(A)
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

```
B <- A^2                       # B[i,j]  = A[i,j]^2,  1 <= i,j <= 2
print(B)
```

```
     [,1] [,2]
[1,]    1    9
[2,]    4   16
```

```
C <- sqrt(B)                   # C[i,j]  = sqrt(A[i,j]^2),  1 <= i,j <= 2
print(C)
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

```
D <- exp(A)                    # D[i,j] = exp(A[i,j]),  1 <= i,j <= 2
print(D)
```

```
         [,1]     [,2]
[1,] 2.718282 20.08554
[2,] 7.389056 54.59815
```

# Binäre arithmetische Funktionen

Matrizen **passender Größen** können mit binären arithmetischen Operatoren verknüpft werden.

**Binäre** arithmetische Operatoren +,-,*,\werden bei gleicher Größe elementweise ausgewertet.

```
A <- matrix(c(1:4), nrow = 2)      # 2 x 2 Matrix der Zahlen 1,2,3,4
print(A)
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

```
B <- matrix(c(5:8), nrow = 2)      # 2 x 2 Matrix der Zahlen 5,6,7,8
print(B)
```

```
     [,1] [,2]
[1,]    5    7
[2,]    6    8
```

```
print(A + B)                       # C[i,j] = A[i,j] + B[i,j], 1 <= i,j <= 2
```

```
     [,1] [,2]
[1,]    6   10
[2,]    8   12
```

```
print(A * B)                       # C[i,j] = A[i,j] * B[i,j], 1 <= i,j <= 2
```

```
     [,1] [,2]
[1,]    5   21
[2,]   12   32
```

## Lineare Algebra mit R Matrizen

- Addition, Subtraktion, Hadamardprodukt elementweise definiert wie oben
- Matrixmultiplikation, Transposition, Inversion, Determinante

```
C <- A  %*% B          # 2 x 2 Matrixprodukt
print(C)
```

```
     [,1] [,2]
[1,]   23   31
[2,]   34   46
```

```
A_T <- t(A)            # Transposition von A
print(A_T)
```

```
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

```
A_inv <- solve(A)      # Inverse von A
print(A_inv)
```

```
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
```

```
A_det <- det(A)        # Determinante von A
print(A_det)
```

```
[1] -2
```

Vektoren

- Übersicht und Erzeugung
- Charakterisierung
- Indizierung
- Arithmetik
- Attribute

**Matrizen**

- Übersicht und Erzeugung
- Charakterisierung
- Indizierung
- Arithmetik
- **Attribute**

# Attribute

Formal sind Matrizen atomare Vektoren mit einem Attribut namens "`dim`".

```
A <- matrix(1:12, nrow = 4 )            #  4 x 3 Matrix
attributes(A)                           # Aufrufen der Attribute von A
```

```
$dim
[1] 4 3
```

`rownames()` und `colnames()` spezifizieren das Attribut "`dimnames`".

```
rownames(A) <- c("P1", "P2", "P3", "P4") # Benennung der Zeilen von A
colnames(A) <- c("Age", "Hgt", "Wgt")    # Benennung der Spalten von A
print(A)                                 # A mit Attribut dimnames
```

```
   Age Hgt Wgt
P1   1   5   9
P2   2   6  10
P3   3   7  11
P4   4   8  12
```

```
attr(A, "dimnames")                      # Aufrufen des Attributs dimnames
```

```
[[1]]
[1] "P1" "P2" "P3" "P4"

[[2]]
[1] "Age" "Hgt" "Wgt"
```

Anmerkung: Bei Matrizen ist die Benennung von Zeilen und Spalten eher ungewöhnlich.