Programmierung und Deskriptive Statistik

BSc Psychologie WiSe 2024/25

Belinda Fleischmann

Datum	Einheit	Thema
11.10.23	Einführung	(1) Einführung
18.10.23	R Grundlagen	(2) R und Visual Studio Code
25.10.23	R Grundlagen	(2) R und Visual Studio Code
01.11.23	**R Grundlagen**	**(3) Vektoren**
08.11.23	R Grundlagen	(4) Matrizen
15.11.23	R Grundlagen	(5) Listen und Dataframes
22.11.23	R Grundlagen	(6) Datenmanagement
29.11.23	Deskriptive Statistik	(7) Häufigkeitsverteilungen
06.12.23	Deskriptive Statistik	(8) Verteilungsfunktionen und Quantile
13.12.23	Deskriptive Statistik	(9) Maße der zentralen Tendenz
20.12.23	Deskriptive Statistik	(10) Maße der Datenvariabilität
20.12.23	*Leistungsnachweis Teil 1*	
	Weihnachtspause	
10.01.24	Deskriptive Statistik	(11) Anwendungsbeispiel (Deskriptive Statistik)
17.01.24	Inferenzstatistik	(12) Anwendungsbeispiel (Parameterschätzung, Konfidenzintervalle)
24.01.24	Inferenzstatistik	(13) Anwendungsbeispiel (Hypothesentest)
25.01.24	*Leistungsnachweis Teil 2*	

(3) Vektoren

Übersicht und Erzeugung

Charakterisierung

Indizierung

Arithmetik

Attribute

Programmierübungen und Selbstkontrollfragen

Übersicht und Erzeugung

Charakterisierung

Indizierung

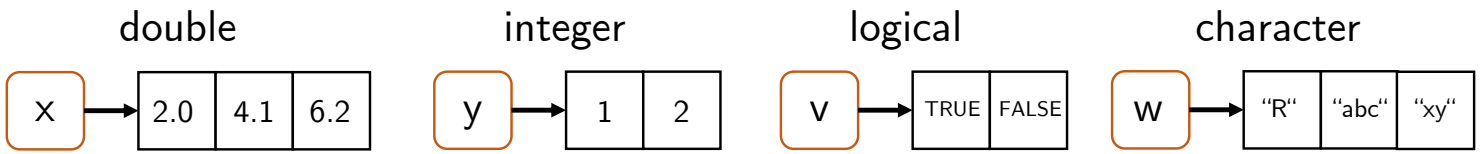
Arithmetik

Attribute

Programmierübungen und Selbstkontrollfragen

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.

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")))`
```

Übersicht und Erzeugung

Charakterisierung

Indizierung

Arithmetik

Attribute

Programmierübungen und Selbstkontrollfragen

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)          # Der 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 der x vom Typ double ist
```

```
[1] FALSE
```

```
is.logical(y)      # Testen, ob der 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
```

Übersicht und Erzeugung

Charakterisierung

Indizierung

Arithmetik

Attribute

Programmierübungen und Selbstkontrollfragen

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"             # Aenderung 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**

```
x <- c(1, 4, 9, 16, 25)   # [1, 4, 9, 16, 25] = [1^2, 2^2, 3^2, 4^2, 5^2]
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**

```
x <- c(1, 4, 9, 16, 25)   # [1, 4, 9, 16, 25] = [1^2, 2^2, 3^2, 4^2, 5^2]
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**

```
x <- c(1, 4, 9, 16, 25)   # [1, 4, 9, 16, 25] = [1^2, 2^2, 3^2, 4^2, 5^2]
y <- x[c(T, T, F, F, T)]  # 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**

```
x <- c(1, 4, 9, 16, 25)   # [1,4,9,16,25] = [1^2, 2^2, 3^2, 4^2, 5^2]
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
```

Übersicht und Erzeugung

Charakterisierung

Indizierung

Arithmetik

Attribute

Programmierübungen und Selbstkontrollfragen

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]
```

- Andernfalls 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)

- Fehlende Werte werden in R mit NA (not applicable) repräsentiert.
- 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()`

```
x <- c(NA, 5, NA, 10)  # Vektor mit NAs
x == NA                # Kein Testen auf NAs : 5 == NA ist NA, nicht FALSE
```

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

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

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

Übersicht und Erzeugung

Charakterisierung

Indizierung

Arithmetik

Attribute

Programmierübungen und Selbstkontrollfragen

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)  # Elementnamengeneration bei Vektorerzeugung
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)
p                            # Vektorausgabe
```

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

Übersicht und Erzeugung

Charakterisierung

Indizierung

Arithmetik

Attribute

Programmierübungen und Selbstkontrollfragen

Programmierübungen

1. Dokumentiere alle in dieser Einheit eingeführten R Befehle in einem R Skript.
2. Öffne die R Help Seite der Funktion `c()`. Wie viele Argumente können der Funktion übergeben werden?
3. Beschreibe in einer Übersicht die R Datenstruktur "Atomarer Vektor".
4. Erläutere die Funktion des Colonoperators in R.
5. Erstelle einen Vektor mit den Zahlen 1, 2, 3 mit dem Colonoperator.
6. Nenne vier Prinzipien der Indizierung in R. Überelege dir zum jedem Prinzip ein Beispiel und dokumentiere es im R Skript aus SKF 1.
7. Erzeuge einen Vektor der Dezimalzahlen 0.0, 0.05, 0.10 , 0.15, ..., 0.90, 0.95, 1.0.
8. Wähle mithilfe positiver Indices die Elemente 0.0, 0.1,..., 0.9, 1.0 dieses Vektors aus.
9. Wähle mithilfe negativer Indizes die Elemente 0.0, 0.1,..., 0.9, 1.0 dieses Vektors aus.
10. Wähle die letzten 10 Elemente dieses Vektors aus.
11. Erläutere, was im Zusammenhang mit der Indizierung in R mit ßu hoher Flexibilität"gemeint ist.

Selbstkontrollfragen

11. Erläutere den Begriff der Datentypangleichung (Coercion).
12. Erläutere den Begriff des (Vektor)Recylings.
13. Erläutere die Bedeutung des R Datentyps `NA`.
14. Erläutere, wofür Attribute in R nützlich sind.