Programmierung und Deskriptive Statistik

BSc Psychologie WiSe 2023/24

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	*Leistungsnachweis Teil 1*	
20.12.23	Deskriptive Statistik	(10) Maße der Datenvariabilität
	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*	

(2) R und VSCode Grundlagen

Getting started

Arithmetik, Logik und Präzedenz

Variablen

Datenstrukturen

Übungen und Selbstkontrollfragen

Getting started

Arithmetik, Logik und Präzedenz

Variablen

Datenstrukturen

Übungen und Selbstkontrollfragen

Installationen

- Die Programmiersprache R
- Die IDE VSCode
- VSCode für R startklar machen (Anleitung hier)
- Optional: radian

Die Basics

- Eingabe von R Befehlen bei >
- Autocomplete mit Tab
- Code ausführen mit Enter
- Vorherige Befehle mit Cursor ↑
- Bereinigen des Konsolenoutputs mit Ctrl + L
- Code Ausführungsstopp mit Esc

Beispiel für einen Befehl

```
print("Hallo Welt!")
```

```
[1] "Hallo Welt!"
```

Anmerkung: **Code-Snippets dieser Form immer aktiv in der Konsole nachvollziehen!**

Neue .R Datei erstellen

- GUI: *File* → *New File* → *R Document (r)*
- Keyboard Shortcut: Ctrl + N

Bestehende .R Datei öffnen

- GUI: File → Open File
- Keyboard Shortcut: Ctrl + O

Code in einem .R Skript schreiben (Befehle in Programmiersprache formulieren)

```
print("Hallo Welt!")  # Hinter Hashtags stehen dokumentierende Kommentare
print("Hallo R!")     # Kommentare werden nicht ausgefuehrt
```

Anmerkung: **Code-Snippets dieser Form immer aktiv in einem R Skript dokumentieren!**

Befehle eines R Scripts ausführen

- Befehle an Console schicken:
  - Einzelnen Zeile, auf welcher der Cursor ruht: Ctrl + Enter
  - Ausführen aller Zeilen im Skript: Ctrl + Shift + Enter
- Ausführen des gesamten .R Skripts (Source):
  - GUI: ▷ - Symbol / Run Source
  - Keyboard Shortcut: Ctrl + Shift + S

Getting started

Arithmetik, Logik und Präzedenz

Variablen

Datenstrukturen

Übungen und Selbstkontrollfragen

R Konsole als Taschenrechner

```r
1 + 1
```

```
[1] 2
```

```r
2 * 3
```

```
[1] 6
```

```r
sqrt(4)
```

```
[1] 2
```

```r
exp(0)
```

```
[1] 1
```

```r
log(1)
```

```
[1] 0
```

Anmerkungen:

- $[1]$ zeigt das erste und einzige Element des Ausgabevektors an
- Vektoren werden noch im Detail behandelt.

Arithmetische Operatoren

Operator	Bedeutung
$+$	Addition
$-$	Subtraktion
$*$	Multiplikation
$/$	Division
$\hat{}$ oder $**$	Potenz
%*%	Matrixmultiplikation
%/%	Ganzzahlige Teilung (5%/%2 = 2)
%%	Modulo (5%%2 = 1)

- Matrixmultiplikation, Modulo, ganzzahlige Teilung benötigen wir zunächst nicht.
- Ganzzahlige Teilung gibt das Resultat der ganzzahligen Teilung an.
- Modulo gibt den ganzzahligen Rest bei ganzzahliger Teilung an.

Logische Operatoren

- Die Boolesche Algebra und R kennen zwei *logische Werte*: TRUE und FALSE
- Bei Auswertung von Relationsoperatoren ergeben sich logische Werte

Relationsoperator	Bedeutung
$==$	Gleich
$!=$	Ungleich
$<$, $>$	Kleiner, Größer
$<=$, $>=$	Kleiner gleich, Größer gleich
$\vert$	ODER
$\&$	UND

- $<, <=, >, >=$ werden zumeist auf numerische Werte angewendet.
- $==, !=$ werden zumeist auf beliebige Datenstrukturen angewendet.
- $\vert$ und $\&$ werden zumeist auf logische Werte angewendet.
- $\vert$ implementiert das inklusive *oder*. Die Funktion xor() implementiert das exklusive ODER.

Mathematische Funktionen

Aufruf	Bedeutung
abs(x)	Betrag
sqrt(x)	Wurzel
ceiling(x)	Aufrunden (ceiling(2.7) = 3)
floor(x)	Abrunden (floor(2.7) = 2)
round(x)	Mathematisches Runden (round(2.5) = 2)
exp(x)	Exponentialfunktion
log(x)	Logarithmus Funktion

- Hierbei handelt es sich um eine Auswahl. Einen vollständigen Überblick gibt

```
names(methods:::.BasicFunsList)
```

- R unterscheidet formal nicht zwischen Operatoren und Funktionen
- Operatoren können mit der Infix Notation als Funktionen genutzt werden

```
`+`(2,3)            # Infixnotation für 2 + 3
```

Operatorpräzedenz (Operatorrangfolge)

- Regeln der Form "Punktrechnung geht vor Strichrechnung".
- Vordefinierte Operatorpräzedenz kann durch Klammern überschrieben werden.

```
2 * 3 + 4
```

```
[1] 10
```

```
2 * (3 + 4)
```

```
[1] 14
```

- Generelle Empfehlung:
  - Operatorrangfolge nicht raten oder folgern, sondern nachschlagen.
  - Lieber Klammern setzen, als keine Klammern setzen.
  - Immer nachschauen, ob Berechnungen die erwarteten Ergebnisse liefern.

Präzedenz und Syntax nachschlagen

```
?Syntax  # öffnet R Help Fenster
```

Operator Syntax and Precedence

Description

Outlines `R` syntax and gives the precedence of operators.

Details

The following unary and binary operators are defined. They are listed in precedence groups, from highest to lowest.

`:: :::`	access variables in a namespace
`$ @`	component / slot extraction
`[ [[`	indexing
`^`	exponentiation (right to left)
`- +`	unary minus and plus
`:`	sequence operator
`%any% |>`	special operators (including `%%` and `%/%`)
`* /`	multiply, divide
`+ -`	(binary) add, subtract
`< > <= >= == !=`	ordering and comparison
`!`	negation
`& &&`	and
`| ||`	or
`~`	as in formulae
`-> ->>`	rightwards assignment
`<- <<-`	assignment (right to left)
`=`	assignment (right to left)
`?`	help (unary and binary)

Within an expression operators of equal precedence are evaluated from left to right except where indicated. (Note that `=` is not necessarily an operator.)

Präzedenz und Ausführungsreihenfolge arithmetischer Operatoren

Operator	Reihenfolge
$\hat{}$	Rechts nach links
$-x, +x$	Unitäres Vorzeichen, links nach rechts
$*$, $/$	Links nach Rechts
$+$, $-$	Links nach Rechts

Beispiele

```
2^2^3          # 2^(2^3)   = 2^8 = 256
(2^2)^3        # (2^2)^3   = 4^3 = 64
-1^2           # -(1^2)    = -1
(-1)^2         # (-1)^2    =  1
2+3/4*5        # 2+(3/4)*5 = 2+(0.75*5) = 2+3.75 = 5.75
2+3/(4*5)      # 2+3/(4*5) = 2+3/20 = 2+0.15 = 2.15
```

Getting started

Arithmetik, Logik und Präzedenz

Variablen

Datenstrukturen

Übungen und Selbstkontrollfragen

Definition

In der Programmierung ist eine Variable ein abstrakter Behälter für eine Größe, welche im Verlauf eines Rechenprozesses auftritt. Im Normalfall wird eine Variable im Quelltext durch einen Namen bezeichnet und hat eine Adresse im Speicher einer Maschine. Der durch eine Variable repräsentierte Wert kann – im Unterschied zu einer Konstante – zur Laufzeit des Rechenprozesses verändert werden.

Wikipedia

- Variablen sind vom Programmierenden benannte Platzhalter für Werte
- In 3GL Sprachen wird der Variablentyp durch eine Initialisierungsanweisung festgelegt:

```
VAR A : INTEGER     # A ist eine Variable vom Typ Integer (ganze Zahl)
```

- In 3GL Sprachen wird Variablen durch eine Zuweisungsanweisung ein Wert zugeschrieben:

```
A := 1              # Der Variable A wird der numerische Wert 1 zugewiesen
```

- In 4GL Sprachen wie Matlab, Python, R werden Variablen durch Zuweisung initialisiert:

```
a = 1               # a ist eine Variable vom Typ double, ihr Wert ist 1
```

- Der Zuweisungsbefehl in Matlab und Python ist =, der Zuweisungsbefehl in R ist <- oder =.
- Offiziell empfohlen für R ist <-.

```
a <- 1              # a ist eine Variable vom Typ double, ihr Wert ist 1
a = 1               # a ist eine Variable vom Typ double, ihr Wert ist 1
```

Beispiel

Greta geht ins Schreibwarengeschäft und kauft vier Hefte, zwei Stifte und einen Füller. Wie viele analoge Gegenstände kauft Greta insgesamt?

Wir definieren zunächst alle Variablen:

```
hefte  <- 4     # Definition der Variable 'hefte'  und Wertzuweisung 4
stifte <- 2     # Definition der Variable 'stifte' und Wertzuweisung 2
fueller <- 1    # Definition der Variable 'fueller' und Wertzuweisung 1
```

Nach Zuweisung existieren die Variablen im Arbeitsspeicher, dem sogenannten *Workspace*. Die Variablen können jetzt wie Zahlen in Berechnungen genutzt werden

```
gesamt <- hefte + stifte + fueller  # Berechnung der Gegenstandsanzahl
print(gesamt)
```

```
[1] 7
```

Ein Heft kostet einen Euro, ein Stift kostet zwei Euro, und ein Füller kostet 10 Euro. Wie viel Euro muss Greta insgesamt bezahlen?

```
gesamtpreis <- hefte * 1 + stifte * 2 + fueller * 10  # Berechung des Preises
print(gesamtpreis)
```

```
[1] 18
```

`print()` gibt Variablenwerte in der R Konsole aus.

Der Workspace | (globaler) Arbeitsbereich

- Enthält alle aktuell definierten Objekte, Variablen und Funktionen während einer R-Sitzung.
- Nach Zuweisung werden Variablen automatisch im Workspace gespeichert.
- Ausgabe aller existierenden benutzbaren Variablen im Arbeitsspeicher in der Konsole.

```
ls()                # Anzeigen aller Variablennamen im aktuellen Workspace
```

```
[1] "fueller"     "gesamt"      "gesamtpreis" "hefte"       "stifte"
```

Löschen von Variablen

- Löschen bestimmter Variablen

```
rm(gesamtpreis)     # Löschen der Variable Gesamtpreis
ls()
```

```
[1] "fueller" "gesamt"  "hefte"   "stifte"
```

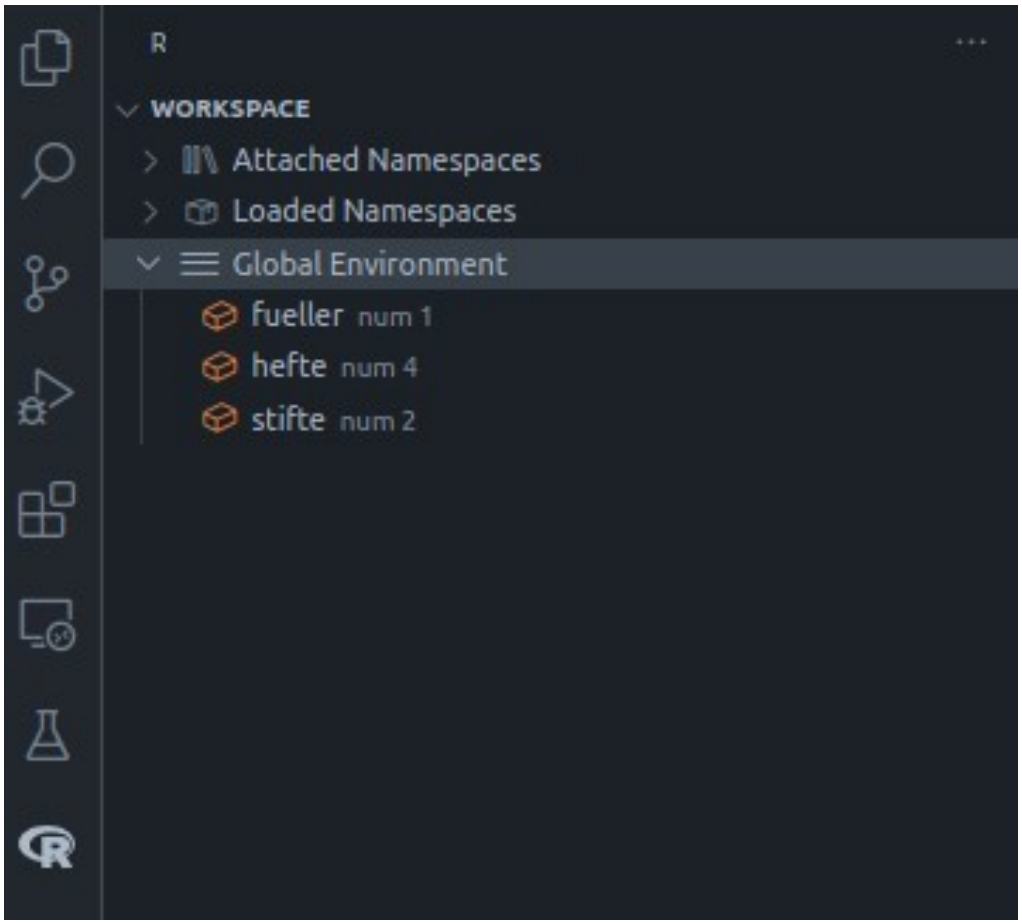
- Löschen aller Variablen im Workspace

```
rm(list = ls())     # Löschen aller Variablen
ls()
```

```
character(0)
```

Workspace

In VSCode kann der Workspace in the R Extension pane angezeigt werden



Variablennamen

Zulässige Variablennamen

- bestehen aus Buchstaben, Zahlen, Punkten (.) und Unterstrichen (_).
- beginnen mit einem Buchstaben oder . nicht gefolgt von einer Zahl.
- dürfen keine *reserverd words* wie for, if, NaN, usw. sein (siehe `?reserved`).
- werden unter `?make.names()` beschrieben.

Sinnvolle Variablennamen

- sind kurz ($\approx$ 1 bis 9 Zeichen) und aussagekräftig.
- bestehen nur aus Kleinbuchstaben und Unterstrichen.

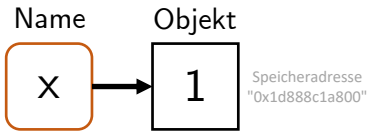
Anmerkung

- R ist *case-sensitive*. Das heißt z.B. x $\neq$ X

Variablenrepräsentation | Binding

```
x <- 1
```

- Intuitiv wird eine Variable genannt x mit dem Wert 1 erzeugt.
- De-facto geschehen zwei Dinge:
  1. R erzeugt ein Objekt (Vektor mit Wert 1) mit Speicheradresse '0x1d888c1a800'.
  2. R verbindet dieses Objekt mit dem Namen x, der das Objekt im Speicher referenziert.

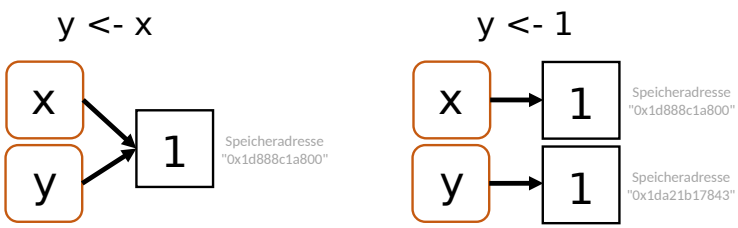



```
y <- x
```

- Intuitiv wird eine Variable genannt y mit Wert gleich dem Wert von x erzeugt.
- De-facto wird ein neuer Name y erzeugt, der dasselbe Objekt referenziert wie x.
- Das Objekt (Vektor mit Wert 1) wird nicht kopiert, R spart Arbeitsspeicher.

```
y <- 1
```

- R erzeugt ein *neues* Objekt (Vektor mit Wert 1) mit *eigener* Speicheradresse '0x1da21b17843'.
- De-facto wird ein neuer Name y erzeugt, der ein anderes Objekt referenziert wie x.



Speicheradressen | object adress

- Speicheraddressen können mit der Funktion `lobstr::obj_addr()` angezeigt werden.

```
library(lobstr)   # Paket `lobstr` laden
x <- 1
obj_addr(x)
```

```
[1] "0x55b3e46c0190"
```

- y bekommt die Zuweisung x. De-facto refenziert y dieselbe Speicheradresse wie x.

```
y <- x
obj_addr(y)
```

```
[1] "0x55b3e46c0190"
```

- y bekommt die Zuweisung zu einem neuen Objekt (Vektor mit Wert 1). De-facto referenziert y eine andere Speicheradresse wie x.

```
y <- 1
obj_addr(x)
```

```
[1] "0x55b3e46c0190"
```

```
obj_addr(y)
```

```
[1] "0x55b3e1c1d9b0"
```

Anmerkungen:

- Ausdrücke der Art `lobstr::obj_addr()` referieren eine Funktion, hier `obj_addr()`, bei gleichzeitiger Angabe des Pakets, hier `lobbstr`.
- Bevor Funktionen eines Pakets verwendet werden können, muss es mit `library()` geladen werden. Ausnahmen sind R Standartpakete, die per default geladen werden, z.B. `base` oder `stats`.

Exkurs: R Pakete

Das Prinzip

- Ein R Paket ist eine Sammlung von R-Funktionen, Datensätzen und vordefinierten Skripten, die in einem strukturierten Format (Paket) gebündelt sind.
- Analog zu Computerprogrammen (wie MS Word) müssen R Pakete zuerst auf dem Rechner *installiert* werden, bevor sie verwendet werden können. Ausnahmen sind Standardpakete (`standard library`), die automatisch mit der Installation von R mitgeliefert werden.
- Analog dazu, dass Computerprogramme zuerst *gestartet* werden müssen, bevor damit gearbeitet werden kann, müssen R Pakete zuerst in die Workspace *geladen* werden, bevor damit gearbeitet werden kann.

Paket-Management

- Anzeige und Auswahl aller installierter Pakete in der R Extention: *HELP PAGES* → *Home* → *Packages*
- Paket-Management mit Befehlen:

```
installed.packages()         # Alle installierten Pakte anzeigen
install.packages("lobstr")   # Paket "lobstr" installieren
library(lobstr)              # Paket "lobstr" laden
```

- Tipp: Wenn in einem Datenanalyseskript Pakete außerhalb der `standard library` verwendet werden, sollte das Laden des Pakets auch ein Befehl im Analyseskript sein!

Variablenrepräsentation | Copy-on-modify

```
x <- 1                          # Objekt (0x74b) erzeugt, x referenziert Speicheradresse des Objektes
y <- x                          # Der Variable y wird der selbe Wert wie x zugewiesen
obj_addr(y) == obj_addr(x)      # Zeigt, dass y dieselbe Speicheradresse (0x74b) wie x referenziert
```

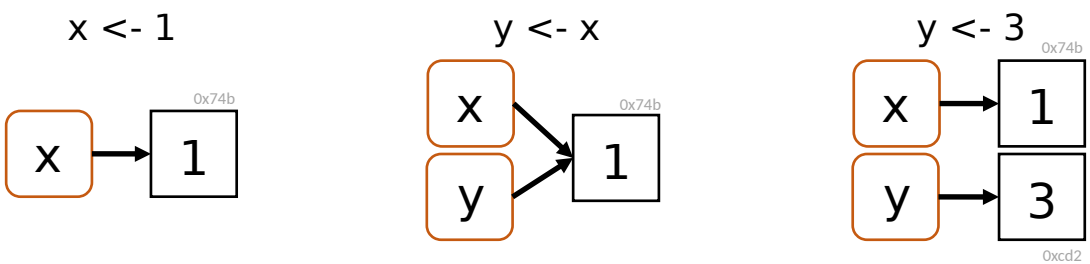
```
[1] TRUE
```

```
y <- 3                          # y modifiziert, modifizierte Kopie (0xcd2) wird gespeichert
obj_addr(y) == obj_addr(x)      # y und x referenzieren jetzt unterschiedliche Speicheradressen
```

```
[1] FALSE
```

```
y <- 1                          # Auch wenn der Wert der Zuweisung gleich ist...
obj_addr(y) == obj_addr(x)      # ...referenzieren y und x nicht die selbe Adresse
```

```
[1] FALSE
```



R Objekte sind *immutable*, können also nicht verändert werden.

Zur Immutability gibt allerdings zwei Ausnahmen, genannt *Modifications-in-place*

1. Objekte mit nur einem gebundenem Namen werden in-place modifiziert

```
x <- c(1,2) # Objekt (0x74b) erzeugt, x referenziert Speicheradresse des Objektes
x[1] <- 2   # Objekt (0x74b) veraendert
```

   - Dieses Verhalten ist allerdings nur in R, nicht innerhalb VSCode reproduzierbar.

2. Environments werden in-place modifiziert ($\rightarrow$ Environments und Funktionen).

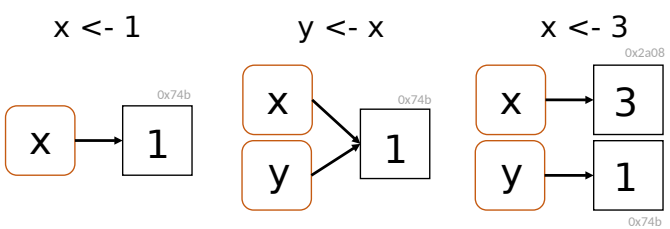
Copy-on-modify gilt auch in umgekehrter Reihenfolge

```
x <- 1                    # Objekt (0x74b) erzeugt, x referenziert Speicheradresse  des Objektes
y <- x                    # Zuweisung y zu gleicher Speicherzelle wie x
obj_addr(y) == obj_addr(x) # zeigt, dass y dieselbe Speicheradresse wie x (0x74b) referenziert.
```

```
[1] TRUE
```

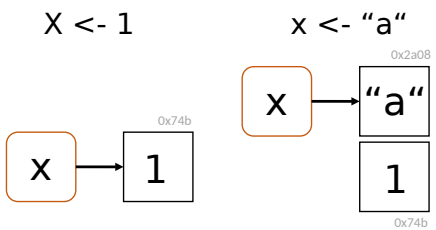
```
x <- 3                    # Ein neues Objekt mit Wert 3 wird in Speicherzelle (0x2a08) erzeugt.
                          # x referenziert jetzt Objelkt (0x2a08)
                          # y referenziert weiterhin Objekt (0x74b)
obj_addr(y) == obj_addr(x) # zeigt, dass y und x nicht mehr dieselbe Speicherzelle addressieren.
```

```
[1] FALSE
```



Unbinding

```
x <- 1      #  x referenziert Objekt (0x74b)
x <- "a"    #  x referenziert Objekt (0x2a08), Objekt (0x74b) jetzt ohne Referenz
```



Garbage collection

- Nicht referenzierte Objekte im Arbeitsspeicher werden automatisch gelöscht.
- Das Löschen geschieht meist erst dann, wenn es wirklich nötig ist.
- Es ist nicht nötig, aktiv die Garbage Collection Funktion `gc()` zu benutzen.

Getting started

Arithmetik, Logik und Präzedenz

Variablen

Datenstrukturen

Übungen und Selbstkontrollfragen

Klassische Datenstrukturen einer 3GL Programmiersprache

Fundamentale Datenstrukturen

- Vordefiniert innerhalb der Programmiersprache
- Logische Werte (`logical`): TRUE, FALSE
- Ganze Zahlen (`integer`): int8 (-128,...,127), int16 (–32768,..., 32767)
- Gleitkommazahlen (single, double): 1.23456, 12.3456, 123.456, ...
- Zeichen (character): “a”, “b”, “c”, “!”
- Datentyp-spezifische assoziierte Operationen
  - AND, OR (logical) +, - (integer) +,-,*, / (single), Zeichenkonkatenation (character)

Zusammengesetzte Datenstrukturen

- Vordefinierte Container zur Zusammenfassung mehrerer Variablen gleichen Datentyps. (z.B. Vektoren, Listen, Arrays, Matrizen, ...)
- Container-spezifische Operationen (z.B. Vektorindizierung, Matrixmultiplikation, ...)

Selbstdefinierte Datenstrukturen

- Definition eigener Datenstrukturen aus vordefinierten Datenstrukturen und Containern
- Definition eigener Operationen

Datenstrukturenkennenlernen beim Erlernen einer Programmiersprache

Fundamentale Datenstrukturen

- Welche fundamentalen Datenstrukturen bietet die Sprache an?
- Welche Operationen darauf sind bereits definiert?
- Wie lautet die Syntax zur Definition einer Variable eines fundamentalen Datentyps?
- Wie lautet die Syntax, um vordefinierte Operationen aufzurufen?

Zusammengesetzte Datenstrukturen

- Welche Container und zugehörige Operationen bietet die Programmiersprache?
- Wie lautet die Syntax zum Umgang mit einem Container?

Selbstdefinierte Datenstrukturen

- Wie erzeugt man selbstdefinierte Datenstrukturen und zugehörige Operationen?
- Wie lautet die Syntax zum Umgang mit einer selbstdefinierten Datenstruktur?

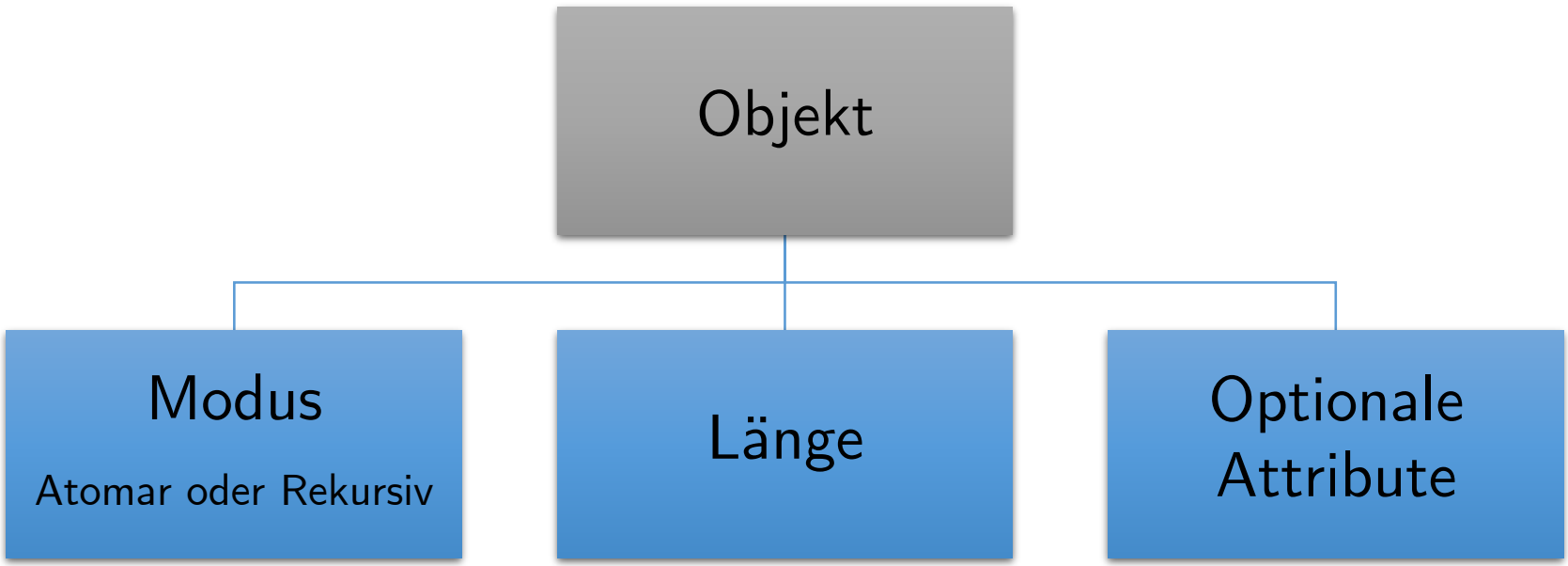
Organisation von Daten in R

Alles, was in R vorkommt, ist ein **Objekt**.

Jedem Objekt kann eindeutig zugeordnet werden:

- ein **Modus**
  - Atomar | Komponenten sind vom gleichen Datentyp.
  - Rekursiv | Komponenten können von unterschiedlichem Datentyp sein.
- eine **Länge**
- optional weitere **Attribute**

Alles, was in R vorkommt, ist ein **Objekt**.



Übersicht der R Datentypen

Datentyp	Erläuterung
logical	Die beiden logischen Werte TRUE und FALSE
double	Gleitkommazahlen
integer	Ganze Zahlen
complex	Komplexe Zahlen, hier nicht weiter besprochen
character	Zeichen und Zeichenketten (strings), 'x' oder "Hallo Welt!"
raw	Bytes, hier nicht weiter besprochen

- Double und integer werden zusammen auch als **numeric** bezeichnet.
- Viele weitere Typen, hier relevant sind **logical**, **double**, **integer**, **character**.

Übersicht der R Datentypen

Automatische Festlegung von Datentypen durch Zuweisung

```
b = TRUE                    # logical
x = 2.5                     # double
y = 1L                      # (long) integer
c = 'a'                     # character
```

Ausgabe von Datentypen durch `typeof()`

```
typeof(b)
```

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

```
typeof(x)
```

```
[1] "double"
```

```
typeof(y)
```

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

```
typeof(c)
```

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

Übersicht der R Datentypen

Testen von Datetypen durch `is.*()`

```
is.logical(x)
```

```
[1] FALSE
```

```
is.double(x)
```

```
[1] TRUE
```

Übersicht atomare Datenstrukturen in R

Datenstruktur	Erläuterung
Vektor	Container von indizierten Komponenten identischen Typs
Matrix	Interpretation eines Vektors als zweidimensionaler Container
Array	Interpretation eines Vektors als mehrdimensionaler Container

$\Rightarrow$ (3) Vektoren und (4) Matrizen

Übersicht rekursive Datenstrukturen in R

Datenstruktur	Erläuterung
Liste	Container von indizierten Komponenten beliebigen Datentyps
	Insbesondere auch rekursive Struktur, z.B. Liste von Listen
Dataframe	Symbiose aus Liste und Matrix
	Jede Komponente ist Vektor beliebigen Datentyps identischer Länge

$\Rightarrow$ (5) Listen und Dataframes

R und RStudio

Arithmetik, Logik und Präzedenz

Variablen

Datenstrukturen

Übungen und Selbstkontrollfragen

Übungen und Selbstkontrollfragen

1. Führen Sie die Befehlssequenz auf Folie R Skripte | Executing and Editing Code aus.
2. Dokumentieren Sie die in dieser Einheit eingeführten R Befehle und Operatoren in einem kommentierten R Skript.
3. Erläutern Sie den Begriff der Operatorpräzedenz.
4. Definieren Sie den Begriff der Variable im Kontext der Programmierung.
5. Erläutern Sie die Begriffe Initialisierungsanweisung und Zuweisungsanweisung für Variablen.
6. Geben Sie jeweils ein Beispiel für einen zulässigen und einen unzulässigen Variablennamen in R.
7. Erläutern Sie den Begriff Workspace.
8. Erläutern Sie die Prozesse, die R im Rahmen einer Zuweisungsanweisung der Form `x <- 1` durchführt.
9. Zeigen Sie, dass zwei Variablen `x` und `y`, die unabhängig voneinander mit dem Wert 7 definiert werden, unterschiedliche Speicheradressen referenzieren.
10. Zeigen Sie, dass zwei Variablen `x` und `y`, die abhängig voneinander mit dem Wert 7 definiert werden, diesselbe Speicheradressen referenzieren.
11. Demonstrieren Sie, dass R case-sensitive ist.
12. Erläutern Sie den Begriffe Copy-on-modify und Modify-in-place.
13. Diskutieren Sie die klassischen Datenstrukturen einer 3GL Programmiersprache.
14. Diskutieren Sie die Organisation von Datenstrukturen in R.
15. Wodurch unterscheiden sich eine atomare und ein rekursive Datenstruktur in R?
16. Nennen und erläutern Sie vier zentrale Datentypen in R.
17. Nennen und erläutern Sie drei zentrale atomare Datenstrukturen in R.
18. Nennen und erläutern Sie zwei zentrale rekursive Datenstrukturen in R.