



# Programmierung und Deskriptive Statistik

BSc Psychologie WiSe 2023/24

Belinda Fleischmann

Inhalte basieren auf Programmierung und Deskriptive Statistik von Dirk Ostwald, lizenziert unter CC BY-NC-SA 4.0

## (2) R und VSCode Grundlagen

R und VSCode

Arithmetik, Logik und Präzedenz

Variablen

Datenstrukturen

Übungen und Selbstkontrollfragen

TODO: Agenda überall einfügen

---

TODO: Agenda einfügen

## R Kommandozeile | Working in the Console

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:

```
::: {.cell}
```

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

```
::: {.cell-output .cell-output-stdout} [1] "Hallo Welt!" ::: :::
```

Anmerkung: **Code-Snippets in diesen Folien immer aktiv in der Konsole nachvollziehen!**

## Neue .R Datei erstellen

- GUI: *File* → *New File* → *R Script*
- Keyboard Shortcut: Ctrl + Shift + N

## Bestehende .R Datei öffnen

- GUI: *File* → *Open File*
- Ctrl + O

## Code im .R Skript schreiben (Befehle in Programmiersprache formulieren)

### Befehle eines R Scripts ausführen

- Einzelnen Zeile, auf welcher der Cursor ruht: ⇒ Run oder Ctrl + Enter
- Ausführen aller Zeilen: ⇒ *Source* oder Ctrl + Shift + Enter oder ⇒ Tickmark bei *Source on Save* setzen und Ctrl + S

Anmerkung: **Code-Snippets in diesen Folien immer aktiv in einem R Skript dokumentieren!**

## Das R und RStudio Data Science Universum

### Analyse & Explore



The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying philosophy and common APIs.

[Project Site Link >](#)



dplyr is the next iteration of plyr, focusing on only data frames. dplyr is faster and has a more consistent API.

[Project GitHub Link >](#)



ggplot2 is an enhanced data visualization package for R. Create stunning multi-layered graphics with ease.

[Project Site Link >](#)



tidyr makes it easy to "tidy" your data. Tidy data is data that's easy to work with: it's easy to munge (with dplyr), visualise (with ggplot2 or ggridis) and model (with R's hundreds of modelling packages).

[Project Paper Link >](#)

### Model & Predict



TensorFlow™ is an open source software library for Machine Intelligence. The R interface to TensorFlow lets you work productively using the High-level Keras and Estimator APIs and the core TensorFlow API.

[Project Site Link >](#)



The tidymodels framework is a collection of packages for modeling and machine learning using tidyverse principles.

[Project Site Link >](#)



Sparklyr provides bindings to Spark's distributed machine learning library. Together with sparklyr's dplyr interface, you can easily create and tune machine learning workflows on Spark, orchestrated entirely within R.

[Project Site Link >](#)

### Connect & Integrate



Sparklyr is an R interface to Apache Spark, a fast and general engine for big data processing. This package connects to local and remote Apache Spark clusters, a dplyr-compatible back end, and an interface to Spark's ML algorithms.

[Project Site Link >](#)



Plumber enables you to convert your existing R code into web APIs by merely adding a couple of special comments.

[Project Site Link >](#)



The reticulate package provides a comprehensive set of tools for interoperability between Python and R.

[Project Site Link >](#)

### Communicate & Interact



Shiny makes it incredibly easy to build interactive web applications with R. Shiny has automatic "reactive" binding between inputs and outputs and extensive pre-built widgets.

[Project Site Link >](#)



Use R Markdown to develop your code and ideas in a reproducible document. Knit plots, tables, and results together with narrative text, and create analyses ready to be shared.

[Project Site Link >](#)

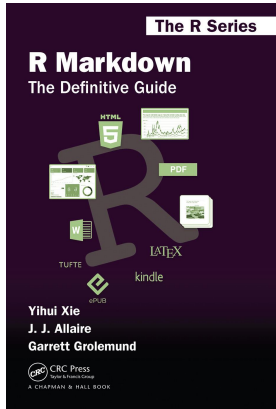


Use flexdashboard to publish groups of related data visualizations on a dashboard.

[Project Site Link >](#)









## Lehrmaterialien mit R und RStudio



## R Kommandozeile | Working in the Console

Die Basics:

- Eingabe von R Befehlen bei >
- Autocomplete mit 
- Code ausführen mit 
- Vorherige Befehle mit Cursor 
- Bereinigen des Konsolenoutputs mit  + 
- Code Ausführungsstopp mit 

Beispiel für einen Befehl:

```
::: {.cell}
```

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

```
::: {.cell-output .cell-output-stdout} [1] "Hallo Welt!" ::: :::
```

Anmerkung:

**Code-Snippets in diesen Folien immer aktiv in der Konsole nachvollziehen!**

## R Skripte | Executing and Editing Code

Neue .R Datei erstellen

- GUI: *File* → *New File* → *R Script*
- Keyboard Shortcut: Ctrl + Shift + N

Bestehende .R Datei öffnen

- GUI: *File* → *Open File*
- Ctrl + O

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

Befehle eines R Scripts ausführen

- Einzelnen Zeile, auf welcher der Cursor ruht: ⇒ Run oder Ctrl + Enter
- Ausführen aller Zeilen: ⇒ *Source* oder Ctrl + Shift + Enter oder ⇒ Tickmark bei *Source on Save* setzen und Ctrl + S

Anmerkung:

**Code-Snippets in diesen Folien immer aktiv in einem R Skript dokumentieren!**

## Das R und RStudio Data Science Universum

### Analyse & Explore



The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying philosophy and common APIs.

[Project Site Link >](#)



dplyr is the next iteration of plyr, focusing on only data frames. dplyr is faster and has a more consistent API.

[Project GitHub Link >](#)



ggplot2 is an enhanced data visualization package for R. Create stunning multi-layered graphics with ease.

[Project Site Link >](#)



tidyr makes it easy to "tidy" your data. Tidy data is data that's easy to work with: it's easy to munge (with dplyr), visualise (with ggplot2 or ggridly) and model (with R's hundreds of modelling packages).

[Project Paper Link >](#)

### Model & Predict



TensorFlow™ is an open source software library for Machine Intelligence. The R interface to TensorFlow lets you work productively using the High-level Keras and Estimator APIs and the core TensorFlow API.

[Project Site Link >](#)



The tidymodels framework is a collection of packages for modeling and machine learning using tidyverse principles.

[Project Site Link >](#)



Sparklyr provides bindings to Spark's distributed machine learning library. Together with sparklyr's dplyr interface, you can easily create and tune machine learning workflows on Spark, orchestrated entirely within R.

[Project Site Link >](#)

### Connect & Integrate



Sparklyr is an R interface to Apache Spark, a fast and general engine for big data processing. This package connects to local and remote Apache Spark clusters, a dplyr-compatible back end, and an interface to Spark's ML algorithms.

[Project Site Link >](#)



Plumber enables you to convert your existing R code into web APIs by merely adding a couple of special comments.

[Project Site Link >](#)



The reticulate package provides a comprehensive set of tools for interoperability between Python and R.

[Project Site Link >](#)

### Communicate & Interact



Shiny makes it incredibly easy to build interactive web applications with R. Shiny has automatic "reactive" binding between inputs and outputs and extensive pre-built widgets.

[Project Site Link >](#)



Use R Markdown to develop your code and ideas in a reproducible document. Knit plots, tables, and results together with narrative text, and create analyses ready to be shared.

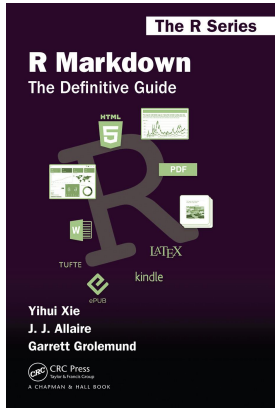
[Project Site Link >](#)



Use flexdashboard to publish groups of related data visualizations on a dashboard.

[Project Site Link >](#)

## Lehrmaterialien mit R und RStudio



---

R und RStudio

## **Arithmetik, Logik und Präzedenz**

Variablen

Datenstrukturen

Übungen und Selbstkontrollfragen

TODO: AGENDA einfügen

```
1 + 1
```

```
[1] 2
```

```
[1] 6
```

```
[1] 1.414214
```

```
[1] 1
```

```
[1] 0
```

Anmerkung:

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



Operator	Bedeutung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
^ 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.

- 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
	ODER
&	UND

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

Aufruf	Bedeutung
<code>abs(x)</code>	Betrag
<code>sqrt(x)</code>	Wurzel
<code>ceiling(x)</code>	Aufrunden ( $\text{ceiling}(2.7) = 3$ )
<code>floor(x)</code>	Abrunden ( $\text{floor}(2.7) = 2$ )
<code>round(x)</code>	Mathematisches Runden ( $\text{round}(2.5) = 2$ )
<code>exp(x)</code>	Exponentialfunktion
<code>log(x)</code>	Logarithmus Funktion

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

```
... {.cell}
```

```
...
```

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

```
... {.cell} ... {.cell-output .cell-output-stdout} [1] 5 ...
```

# Operatorpräzedenz

- Operatorrangfolge
- Regeln der Form “Punktrechnung geht vor Strichrechnung”
- Vordefinierte Operatorpräzedenz kann durch Klammern überschrieben werden

```
::: {.cell} ::: {.cell-output .cell-output-stdout} [1] 10 ::: :::
```

```
::: {.cell}  
::: {.cell-output .cell-output-stdout}  
:::
```

```
[1] 14
```

```
:::
```

```
:::
```

```
:::
```

## Generell gilt

- Operatorrangfolge nicht raten oder folgern, sondern nachschauen!
- Lieber Klammern setzen, als keine Klammern setzen!
- Immer nachschauen, ob Berechnungen die erwarteten Ergebnisse liefern!

```
::: {.cell}
```

```
:::
```

## Präzedenz und Ausführungsreihenfolge arithmetischer Operatoren

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

### Beispiele

# Operatorpräzedenz

## 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 %x% 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.)

TODO: Agenda einfügen

## 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- Grundlagen

- 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:

```
::: {.cell}
```

```
:::
```

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

```
::: {.cell}
```

```
:::
```

- Der Zuweisungsbefehl in Matlab und Python ist =, der Zuweisungsbefehl in R ist <- oder =.
- Offiziell empfohlen für R ist <-, aus Kohärenzgründen benutzen wir hier =.

## Variablen - 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:

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

```
[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?

```
[1] 18
```

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

Nach Zuweisung werden Variablen automatisch im Workspace gespeichert.

- In der RStudio IDE wird das im *Environment* Pane angezeigt.
- `ls()` zeigt die existierenden benutzbaren Variablen im Arbeitsspeicher an

```
::: {.cell} ::: {.cell-output .cell-output-stdout} [1] äbb_dir"          fueller"          "gesamt"  
"gesamtpreis"      [5] "has_annotations" "hefte"          ßtifte" ::: :::
```

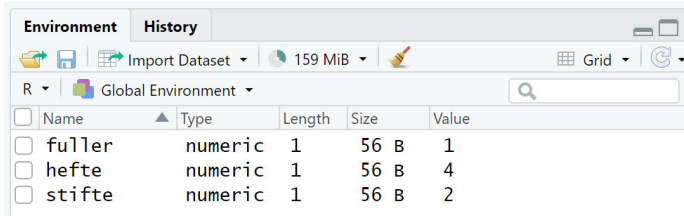
## Löschen von Variablen

- `rm()` erlaubt das Löschen von Variablen

```
::: {.cell} ::: {.cell-output .cell-output-stdout} [1] äbb_dir"          fueller"          "gesamt"  
"has_annotations" [5] "hefte"          ßtifte" ::: :::
```

- `rm(list=ls())` löscht alle Variablen

```
::: {.cell} ::: {.cell-output .cell-output-stdout} character(0) ::: :::
```



The screenshot shows the R Environment pane with the 'Global Environment' selected. It displays three variables: 'fuller', 'hefte', and 'stifte', all of type 'numeric'. The 'fuller' variable has a length of 1 and a size of 56 B, with a value of 1. The 'hefte' variable has a length of 1 and a size of 56 B, with a value of 4. The 'stifte' variable has a length of 1 and a size of 56 B, with a value of 2. The pane also includes tabs for 'Environment' and 'History', a search bar, and various icons for file operations and memory management.

<input type="checkbox"/>	Name	Type	Length	Size	Value
<input type="checkbox"/>	fuller	numeric	1	56 B	1
<input type="checkbox"/>	hefte	numeric	1	56 B	4
<input type="checkbox"/>	stifte	numeric	1	56 B	2

# Variablennamen

---

## Variablennamen

### Zulässige Variablennamen

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

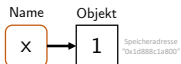
### Sinnvoll Variablennamen

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

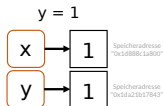
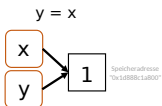
### Anmerkung

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

- 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.



- 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.
- 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 können mit `lobstr::obj_addr()` angezeigt werden.  
::: {.cell} ::: {.cell-output .cell-output-stdout} [1] "0x556382edbce8" ::: :::
- `y` bekommt die Zuweisung `x`. De-facto referenziert `y` dieselbe Speicheradresse wie `x`.  
::: {.cell} ::: {.cell-output .cell-output-stdout} [1] "0x556382edbce8" ::: :::
- `y` bekommt die Zuweisung zu einem neuen Objekt (Vektor mit Wert 1). De-facto referenziert `y` eine andere Speicheradresse wie `x`.  
::: {.cell} ::: {.cell-output .cell-output-stdout} [1] "0x5563834fe6d0" ::: :::

### Anmerkungen:

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

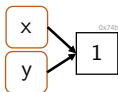
[1] 3

[1] 1

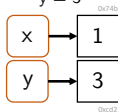
x = 1



y = x



y = 3



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

... {.cell}

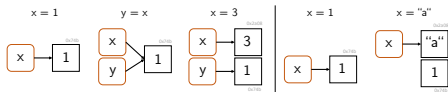
...

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

2. Environments werden in-place modifiziert (→ Environments und Funktionen).

Copy-on-modify gilt auch in umgekehrter Reihenfolge

[1] 1



Unbinding

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.



## 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

## 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

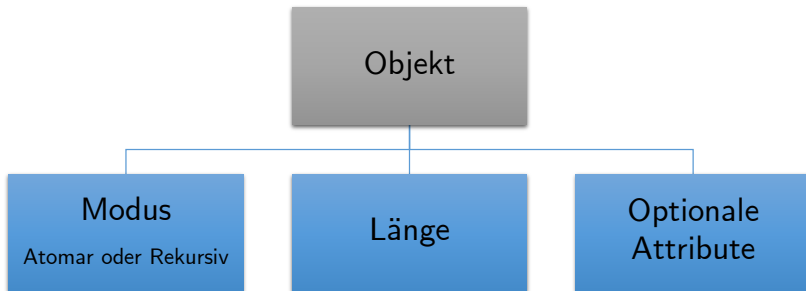
- Wie erzeugt man selbstdefinierte Datenstrukturen und zugehörige Operationen?
- Wie lautet die Syntax zum Umgang mit einer selbstdefinierten Datenstruktur?

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

Testen von Datentypen durch `typeof()`

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

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

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

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

Testen von Datentypen durch `is.*()`

```
[1] FALSE
```

```
[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

⇒ (3) Vektoren, Matrizen, Arrays

# 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

⇒ (4) Listen und Dataframes

R und RStudio

Arithmetik, Logik und Präzedenz

Variablen

Datenstrukturen

**Übungen und Selbstkontrollfragen**

# Übungen und Selbstkontrollfragen

---

1. Installieren Sie R und RStudio auf Ihrem Rechner.
2. Führen Sie die Befehlssequenz auf Folie [R Skripte | Executing and Editing Code](#) aus.
3. Dokumentieren Sie die in dieser Einheit eingeführten R Befehle in einem kommentierten R Skript.
4. Erläutern Sie den Begriff der Operatorpräzedenz.
5. Definieren Sie den Begriff der Variable im Kontext der Programmierung.
6. Erläutern Sie die Begriffe Initialisierungsanweisung und Zuweisungsanweisung für Variablen.
7. Erläutern Sie den Begriff Workspace.
8. Geben Sie jeweils ein Beispiel für einen zulässigen und einen unzulässigen Variablennamen in R.
9. Erläutern Sie die Prozesse, die R im Rahmen einer Zuweisungsanweisung der Form  $x = 1$  durchführt.
10. Erläutern Sie den Begriffe Copy-on-modify und Modify-in-place.
11. Diskutieren Sie die klassischen Datenstrukturen einer 3GL Programmiersprache.
12. Diskutieren Sie die Organisation von Datenstrukturen in R.
13. Wodurch unterscheiden sich eine atomare und ein rekursive Datenstruktur in R?
14. Nennen und erläutern Sie vier zentrale Datentypen in R.
15. Nennen und erläutern Sie vier zentrale atomare Datenstrukturen in R.
16. Nennen und erläutern Sie zwei zentrale rekursive Datenstrukturen in R.

## References

---