

02_einfuehrung_in_r

Erste Schritte, Dateneingabe, Datenzugriff,
Einlesen und Speichern



R starten

Starten des Systems:

Sie landen auf dem Prompt, der Eingabeaufforderung. Ggf. wird eine vorher gespeicherte Arbeitsumgebung geladen

```
>
```

Ändern Sie das Arbeitsverzeichnis:

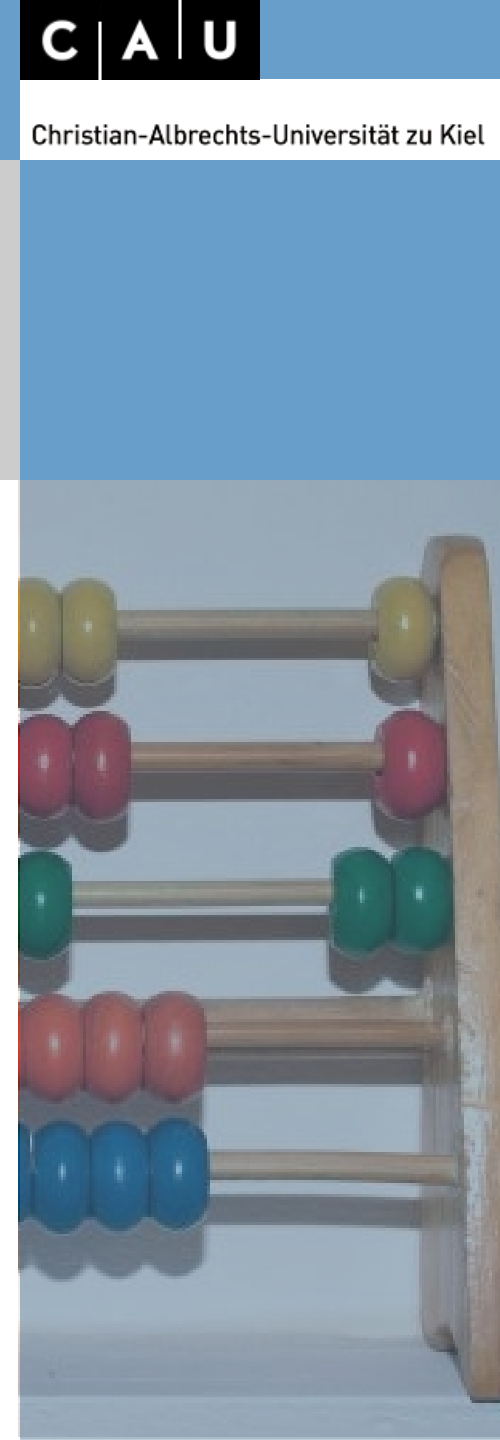
```
> getwd()  
[1] "/home/martin" # oder etwas anderes...  
> setwd("U:\\R")
```

Pfad anpassen

Graphische Benutzeroberfläche:

R-Commander

```
> library(Rcmdr)
```



R als Taschenrechner benutzen

Einfachste Nutzungsmöglichkeit:

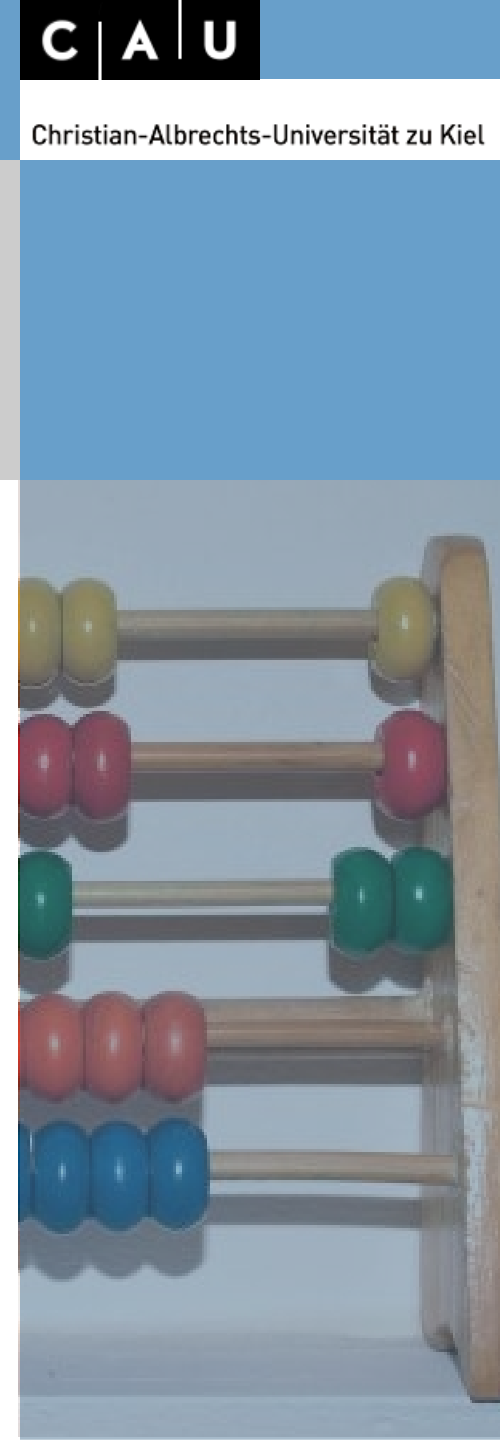
```
> 2+2  
[1] 4  
> 2^2  
[1] 4
```

Mehrere aufeinander folgende Kommandos durch ; abgrenzen:

```
> (1-2)*3; 1-2*3  
[1] -3  
[1] -5
```

Funktionen nutzen:

```
> sqrt(2)           #Wurzel ziehen  
[1] 1.414214  
> log(10)           #Logarithmus Basis e  
[1] 2.303  
> log(10, 10)       #Logarithmus Basis 10, wie log(10,  
base=10)  
[1] 1
```



Hilfe in R

Aufruf der Hilfefunktionen:

```
> help(sqrt)
```

...

Verlassen mit Eingabe `q`

Noch einfacher:

```
> ? sqrt
```

Suchen von Hilfeseiten:

```
> help.search("logarithm")
```

...

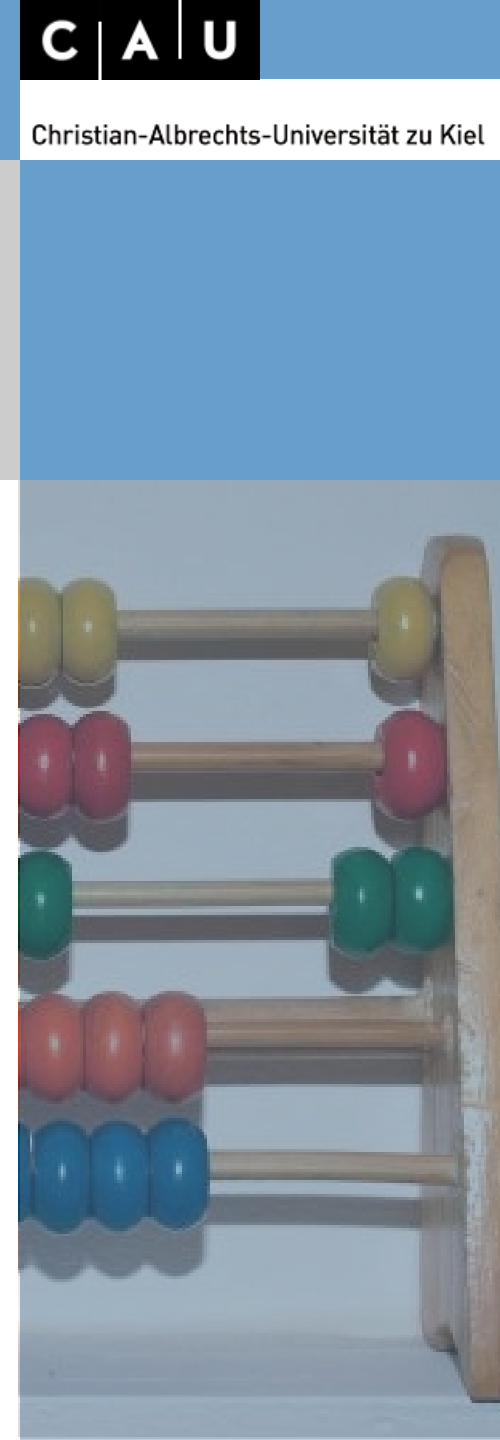
Aufruf der Hilfeseiten im HTML-Format:

```
> help.start()
```

...

Zurück zur normalen Hilfedarstellung:

```
> options(htmlhelp = FALSE)
```



Zuweisung von Daten zu Variablen

Benennen von Variablen für Werte (Zuweisung):

```
> x<-2      #Es wird keine Meldung zurückgegeben
> x
[1] 2
> pi        #Eingebaute Variable
[1] 3.141593
```

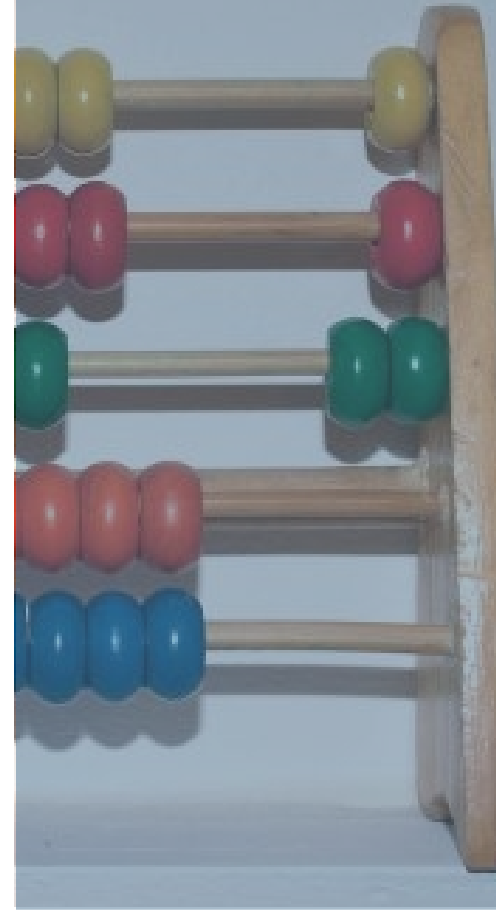
Pfeil oder Gleichheitszeichen?

Zuweisung klassisch bei R:

```
> x=2      #Es wird keine Meldung zurückgegeben
> x
[1] 2
```

Beides in neueren möglich, Frage des Geschmacks

<- ist klarer, wird hier benutzt



Arbeiten mit Variablen

Anzeigen der bisher vergebenen Variablen:

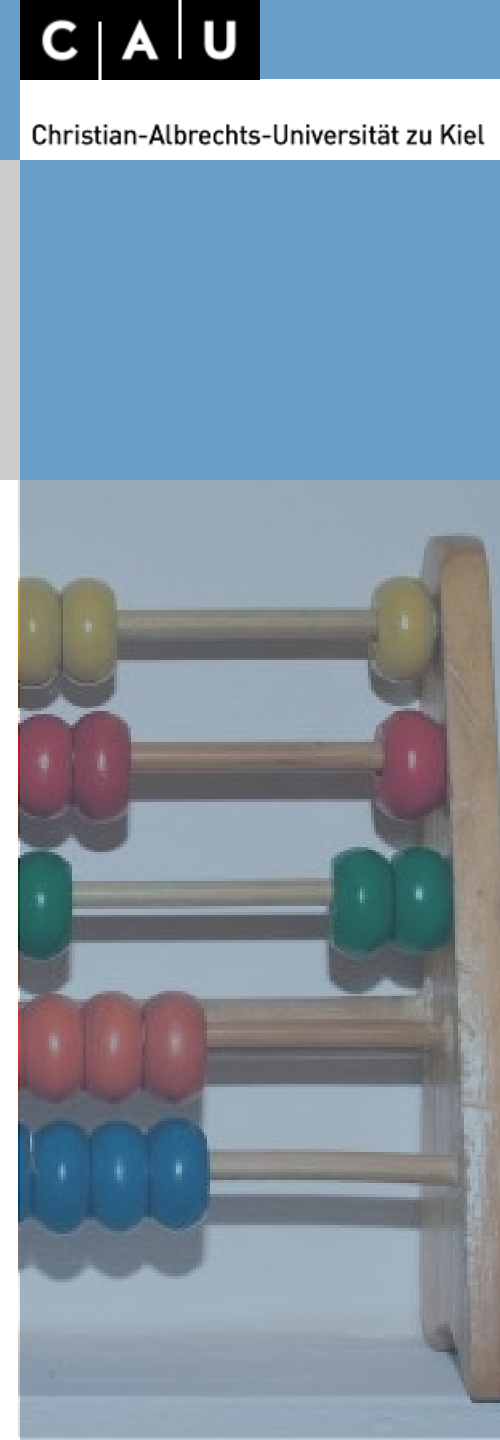
```
> ls()  
[1] "x"
```

Löschen einer Variable:

```
> rm(x)      #Es wird keine Meldung zurückgegeben  
> ls()  
[1] character(0)
```

Rechnen mit Variablen:

```
> x<-2        #Es wird keine Meldung zurückgegeben  
> y<-2*x      #Es wird keine Meldung zurückgegeben  
> z<-sqrt(x)  #Es wird keine Meldung zurückgegeben  
> ls()  
[1] "x" "y" "z"  
> y  
[1] 4  
> z  
[1] 1.414214
```

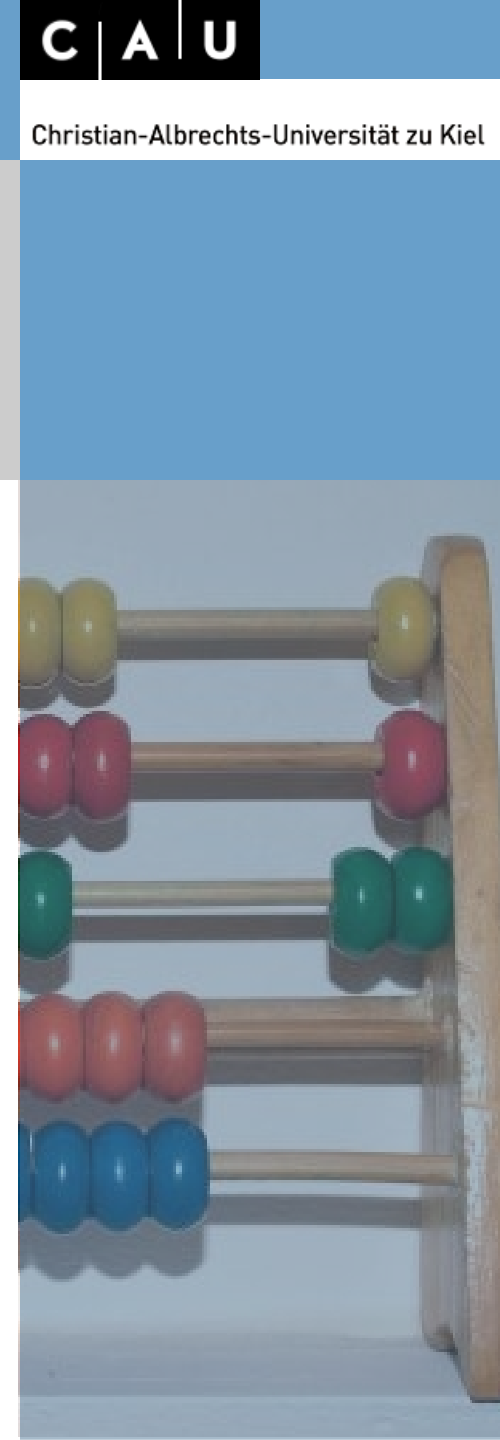


Aufgabe Variablen

Kreisberechnung:

Berechnen Sie den Durchmesser d eines Kreise mit dem Radius $r=5$, den u Umfang ($2\pi r$) und dessen A Flächeninhalt (πr^2)

Addieren Sie Flächeninhalt und Umfang, weisen Sie das Ergebnis der Variable v zu und löschen Sie u und A .



Aufgabe Variablen

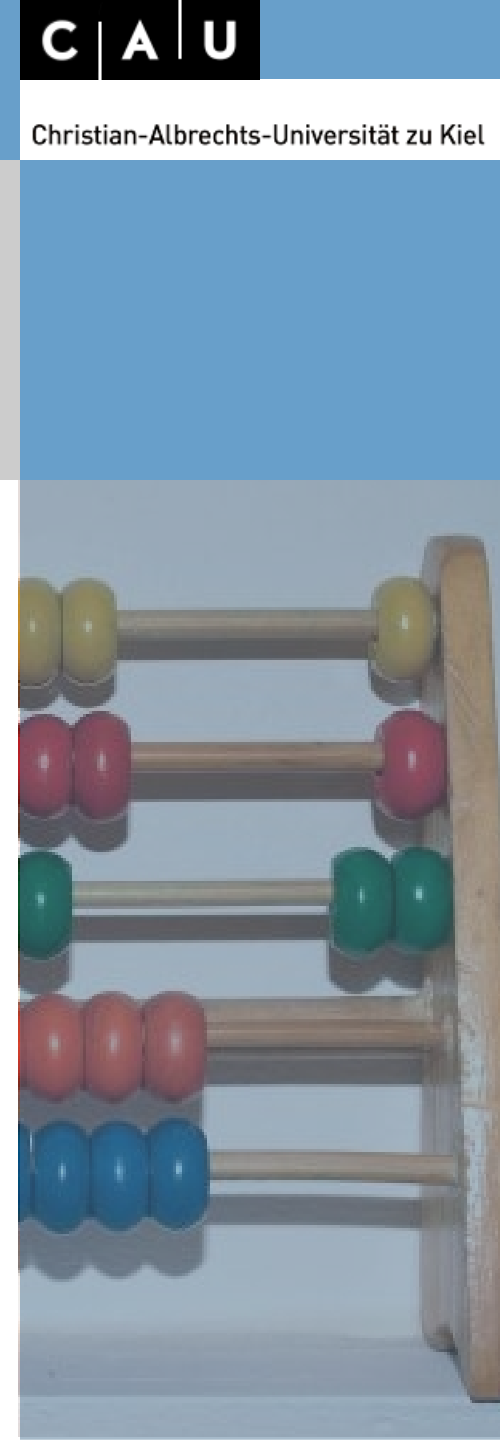
Kreisberechnung:

Berechnen Sie den Durchmesser d eines Kreise mit dem Radius $r=5$, den u Umfang ($2\pi r$) und dessen A Flächeninhalt (πr^2)

Addieren Sie Flächeninhalt und Umfang, weisen Sie das Ergebnis der Variable v zu und löschen Sie u und A .

Ergebnis:

```
> ls()  
[1] "d" "r" "v" "x" "y" "z"  
> v  
[1] 109.9557  
>
```



Skalare, Vektoren, Matrizen, Dataframes

Skalar:

Eine einzelne Zahl oder ein einzelnes Datum

```
> pi  
[1] 3.141593
```

Vektor:

Eine Reihe von Zahlen oder Daten

```
> ls()  
[1] "d" "r" "v" "x" "y" "z"
```

Matrix:

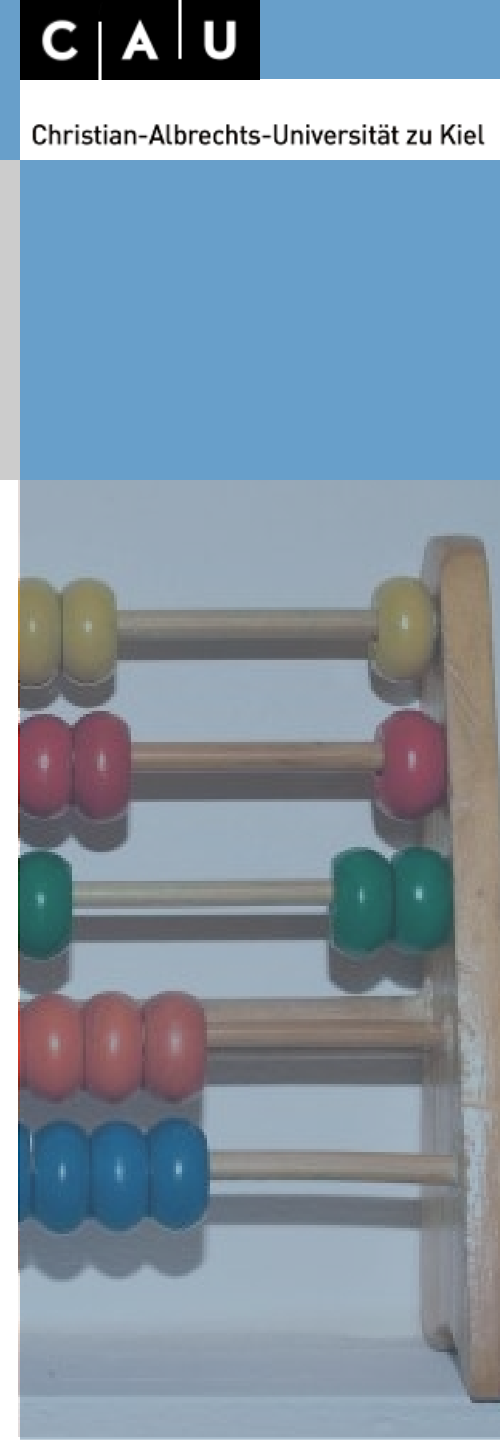
Eine Datentabelle mit Daten gleicher Art

```
> euro.cross  
...
```

Dataframe:

Eine Datentabelle mit Daten unterschiedlicher Art

```
> mtcars  
...
```



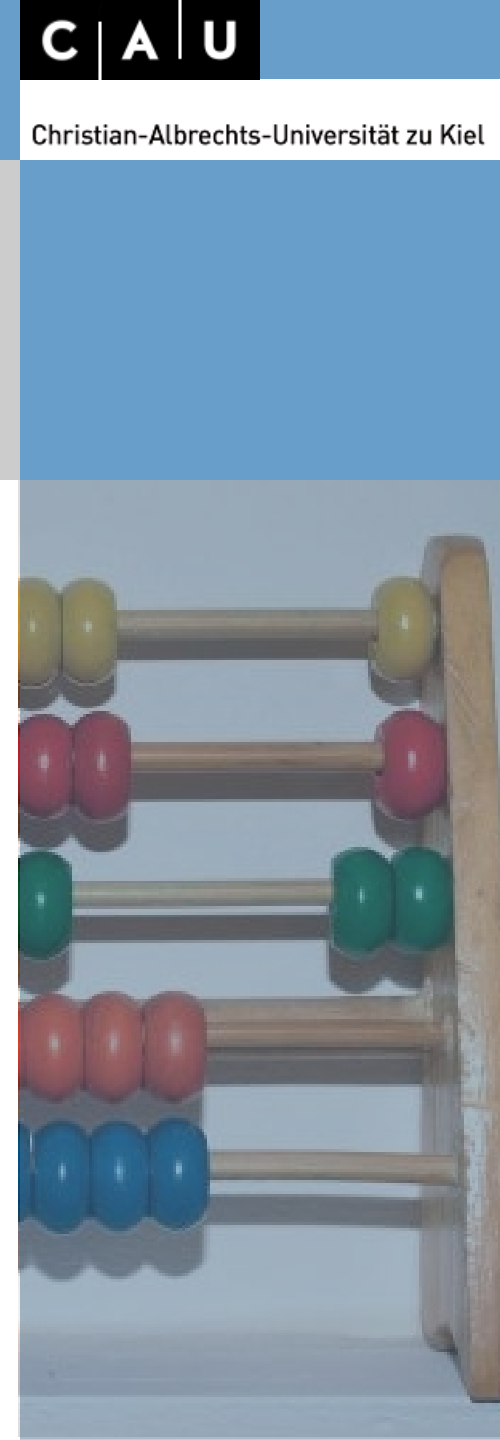
c() für Dateneingabe benutzen

Eingabe von Werten in einen Vektor:

```
> Fundorte <- c("Leubingen", "Melz", "Bruszczewo")
> Fundorte
[1] "Leubingen" "Melz"      "Bruszczewo"
> Fundkategorien <- c("Grab", "Hort", "Siedlung")
> Fundkategorien
[1] "Grab"      "Hort"      "Siedlung"
> c(Fundorte, Fundkategorien)
[1] "Leubingen" "Melz"      "Bruszczewo" "Grab"
  "Hort"
[6] "Siedlung"
```

Datenvektoren benennen:

```
> names(Fundorte) <- Fundkategorien
> Fundorte
      Grab      Hort      Siedlung
"Leubingen"  "Melz" "Bruszczewo"
```



Funktionen auf Vektoren anwenden

Daten:

```
kursteilnehmer<-c("Ria", "Anja", "Hannes", "Moritz",  
"Basti", "Kay", "Björn", "Cristin", "Martin")  
koerpergroessen<-c(174, 163, 182, 175, 173, 198, 179,  
163, 181)  
names(koerpergroessen)<-kursteilnehmer
```

Summe:

```
> sum(koerpergroessen)  
[1] 1588
```

Anzahl:

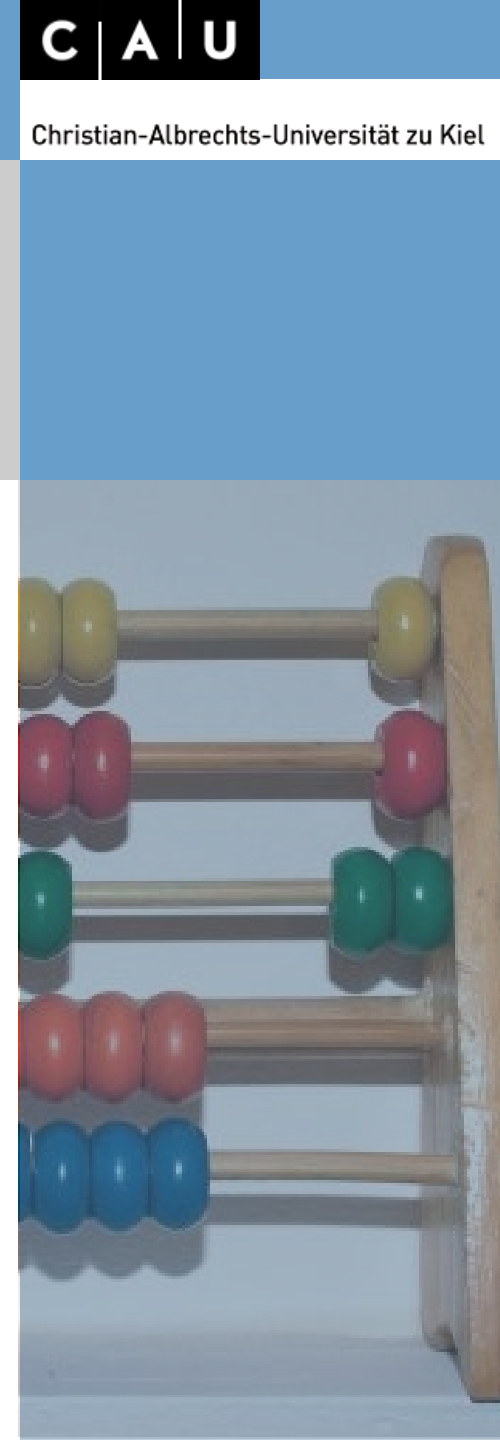
```
> length(koerpergroessen)  
[1] 9
```

Arithm. Mittel:

```
> sum(koerpergroessen)/length(koerpergroessen)  
[1] 176.4444
```

Oder bequemer:

```
> mean(koerpergroessen)  
[1] 176.4444
```



Funktionen auf Vektoren anwenden 2

sortieren:

```
> sort(koerpergroessen)
```

Anja	Cristin	Basti	Ria	Moritz	Björn
Martin	Hannes	Kay			
163	163	173	174	175	179
181	182	198			

Kleinster Wert:

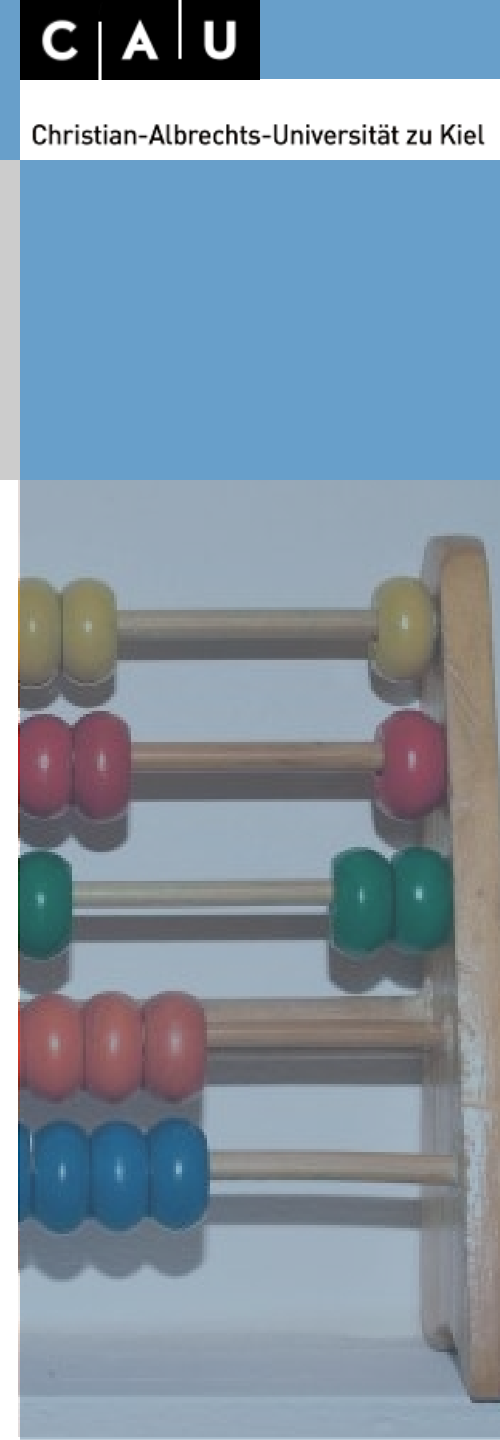
```
> min(koerpergroessen)
[1] 163
```

Größter Wert:

```
> max(koerpergroessen)
[1] 198
```

Oder bequemer:

```
> range(koerpergroessen)
[1] 163 198
```



Funktionen auf Vektoren anwenden 3

Verändern der Werte durch Berechnung:

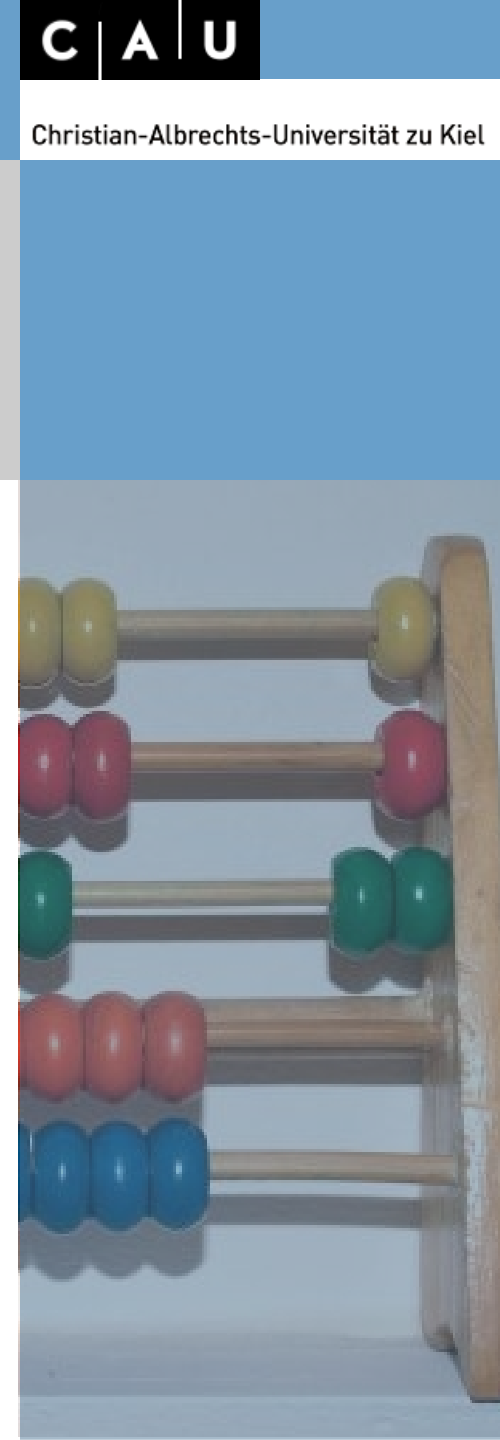
```
> koerpergroessen.in.m <- koerpergroessen/100
> koerpergroessen.in.m
```

	Ria	Anja	Hannes	Moritz	Basti	Kay
Björn	Cristin	Martin				
	1.74	1.63	1.82	1.75	1.73	1.98
	1.79	1.63	1.81			

Aber:

```
> test<-c(1,2,3,4,5,6,7,8,9)
> koerpergroessen.in.m + test
```

	Ria	Anja	Hannes	Moritz	Basti	Kay
Björn	Cristin	Martin				
	2.74	3.63	4.82	5.75	6.73	7.98
	8.79	9.63	10.81			



Aufgabe Variablen

Keramikaufnahme:

Für eine Grabung sind folgende Anzahlen von Flintartefakten bekannt:

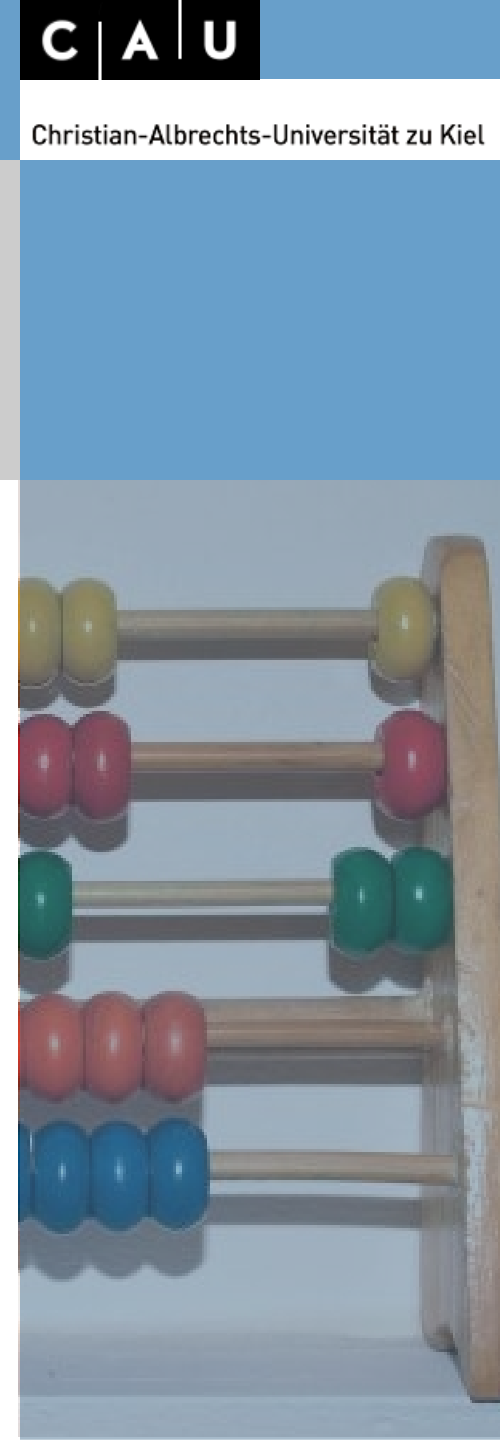
Abschläge	Klingen	Kerne	Trümmer
506	104	30	267

Erstellen Sie einen benannten Vektor, berechnen Sie die durchschnittliche Anzahl und sortieren Sie den Vektor nach Anzahl

Eine Kiste fehlte bei der Aufnahme, folgende Werte kommen hinzu:

Abschläge	Klingen	Kerne	Trümmer
52	24	15	83

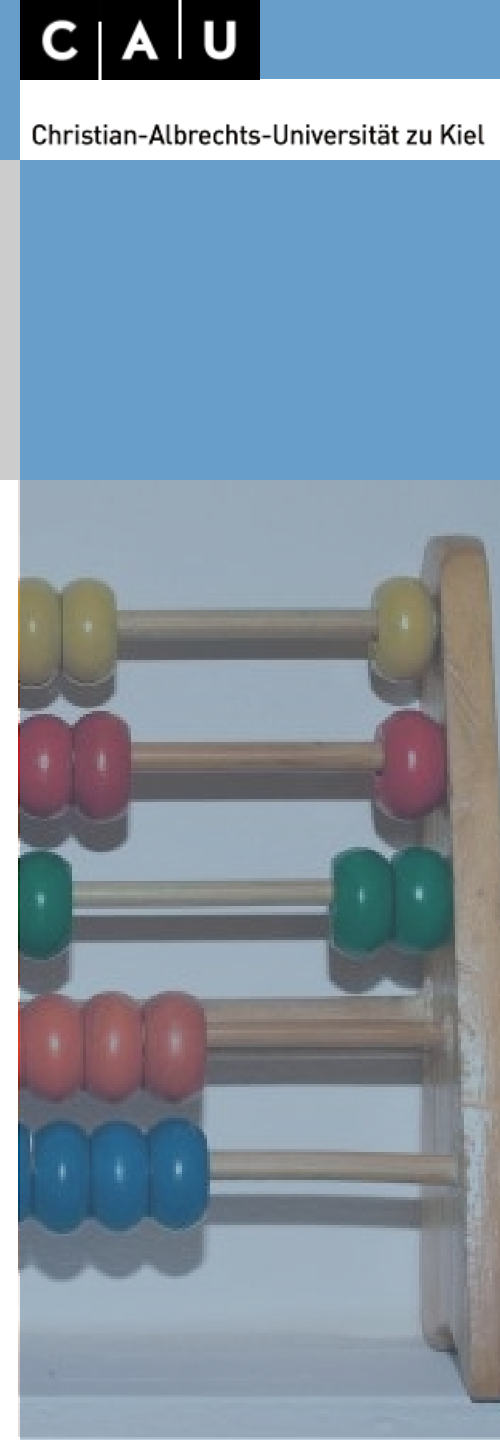
Zudem wurden bei jedem der Artefakttypen 10 Einheiten unterschlagen. Erstellen Sie einen Vektor für die fehlende Kiste, addieren Sie diesen und die fehlenden 10 dazu und wiederholen Sie die Berechnung



Aufgabe Variablen

Keramikaufnahme:

```
> ww1<-c(506,104,30,267) #Werte eingeben
> names(ww1)<-
c("Abschlaege","Klingen","Kerne","Trümmer") #Namen
> ww1.prozent<-ww1/sum(ww1) #Anteile berechnen
> sort(ww1.prozent) #sortiert ausgeben
      Kerne      Klingen      Trümmer Abschlaege
0.03307607 0.11466373 0.29437707 0.55788313
> ww2<-c(52,24,15,83) #fehlende Kiste eingeben
> ww3<-ww1+ww2 #fehlende Kiste hinzuaddieren
> ww3<-ww3+10 #10 zu allen Werten hinzuaddieren
> ww3.prozent<-ww3/sum(ww3) #Anteile berechnen
> sort(ww3.prozent) #sortiert ausgeben
      Kerne      Klingen      Trümmer Abschlaege
0.04906334 0.12310437 0.32114184 0.50669045
```



Sequenzen und wiederholte Daten erzeugen

Einfache Sequenz:

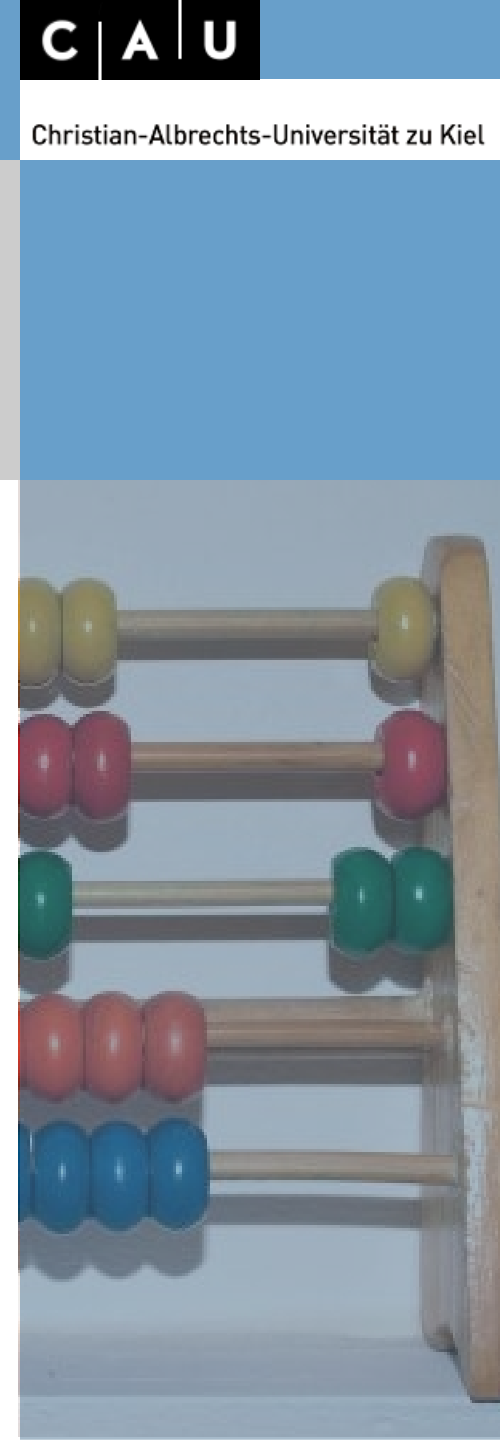
```
> 1:10  
[1] 1 2 3 4 5 6 7 8 9 10
```

Sequenz mit Start- und Endpunkt sowie Schrittweite:

```
> seq(1,10,by=2)  
[1] 1 3 5 7 9  
  
> seq(1,20,length=5)  
[1] 1.00 5.75 10.50 15.25 20.00
```

Wiederholte Daten:

```
> rep(1,10)  
[1] 1 1 1 1 1 1 1 1 1 1  
> rep(1:3,3)  
[1] 1 2 3 1 2 3 1 2 3  
> rep(c("Anton","Berta","Claudius"),3)  
[1] "Anton" "Berta" "Claudius" "Anton"  
"Berta" "Claudius"  
[7] "Anton" "Berta" "Claudius"
```



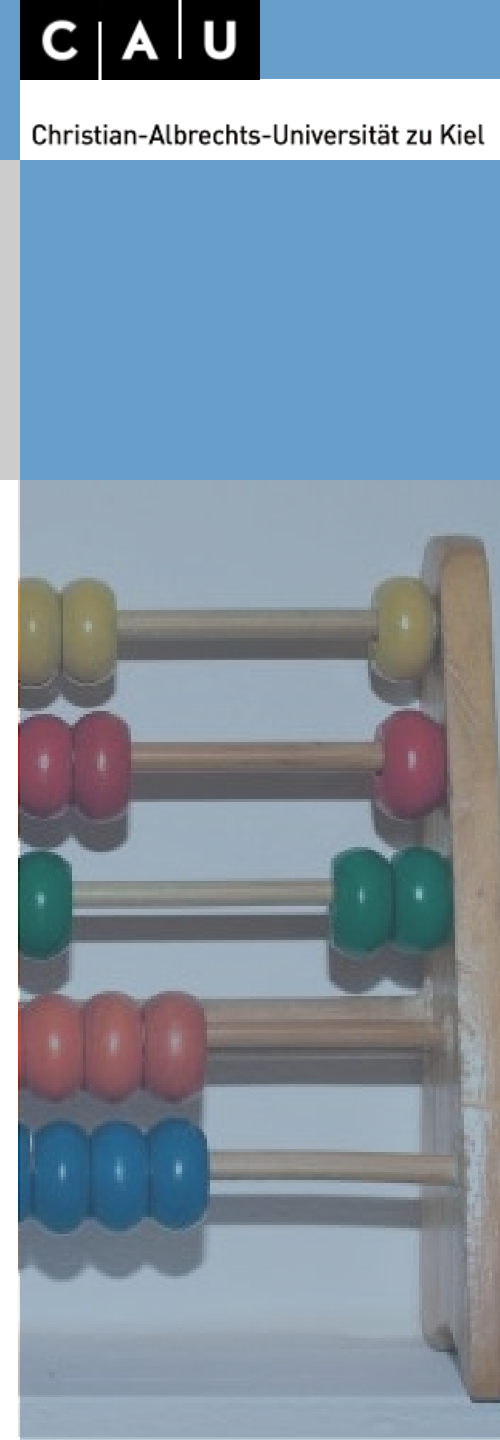
Datenzugriff durch Indizes

Zugriff durch Position:

```
> koerpergroessen[1]
Ria
174
> koerpergroessen[5]
Basti
173
> koerpergroessen[1:3]
  Ria   Anja Hannes
174   163   182
> koerpergroessen[-(1:3)]
Moritz   Basti   Kay   Björn Cristin   Martin
175      173    198   179      163     181
```

Zugriff durch Name:

```
> koerpergroessen["Kay"]
Kay
198
```



Dateneingabe in Vektoren

Eingabe nach Position:

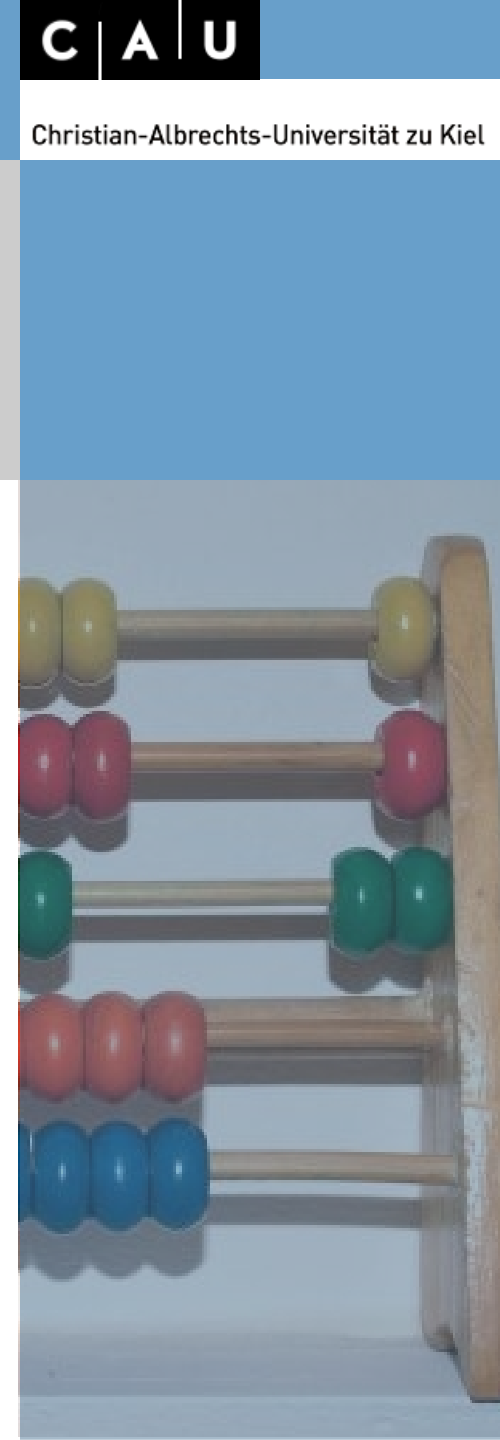
```
> ww1
Abschlaege      Klingen      Kerne      Trümmer
          506          104          30          267
> ww1[1]<-483
> ww1[1]
Abschlaege
          483
```

Eingabe nach Namen:

```
> ww1["Kerne"]<-26
> ww1
Abschlaege      Klingen      Kerne      Trümmer
          483          104          26          267
```

Recycling:

```
> ww1[1:length(ww1)]<-c(30,50)
> ww1
Abschlaege      Klingen      Kerne      Trümmer
          30          50          30          50
```



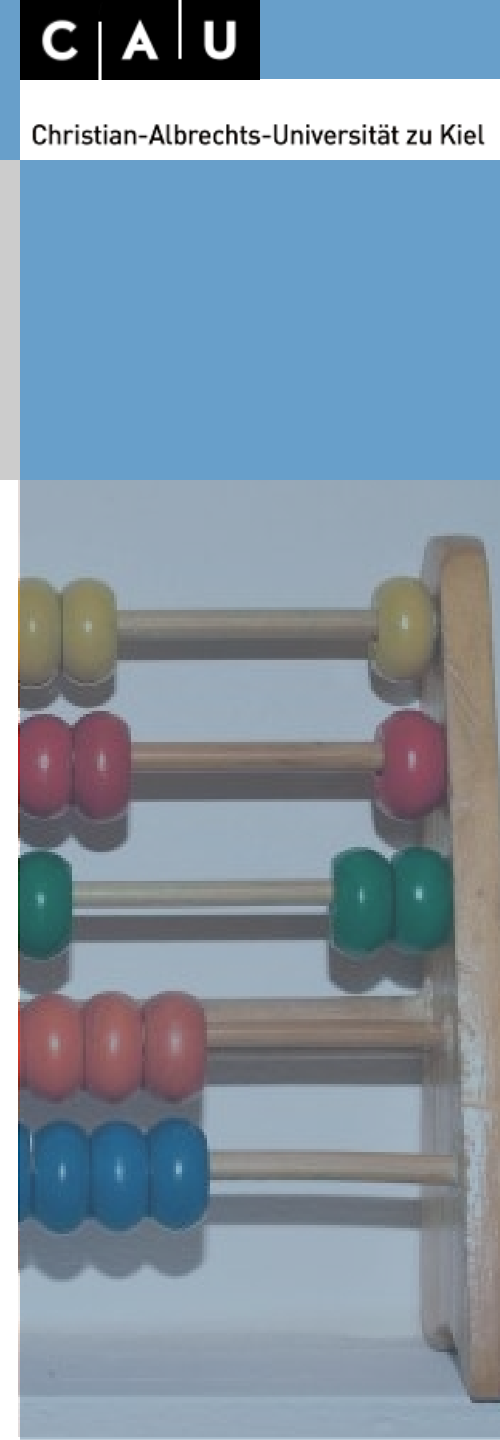
Logische Werte

Ja/Nein-Werte:

```
> pi>4
[1] FALSE
> koerpergroessen > 175
      Ria      Anja  Hannes  Moritz   Basti      Kay
Björn Cristin  Martin
  FALSE   FALSE   TRUE   FALSE   FALSE   TRUE
TRUE   FALSE   TRUE
```

Können zum Auswählen verwendet werden:

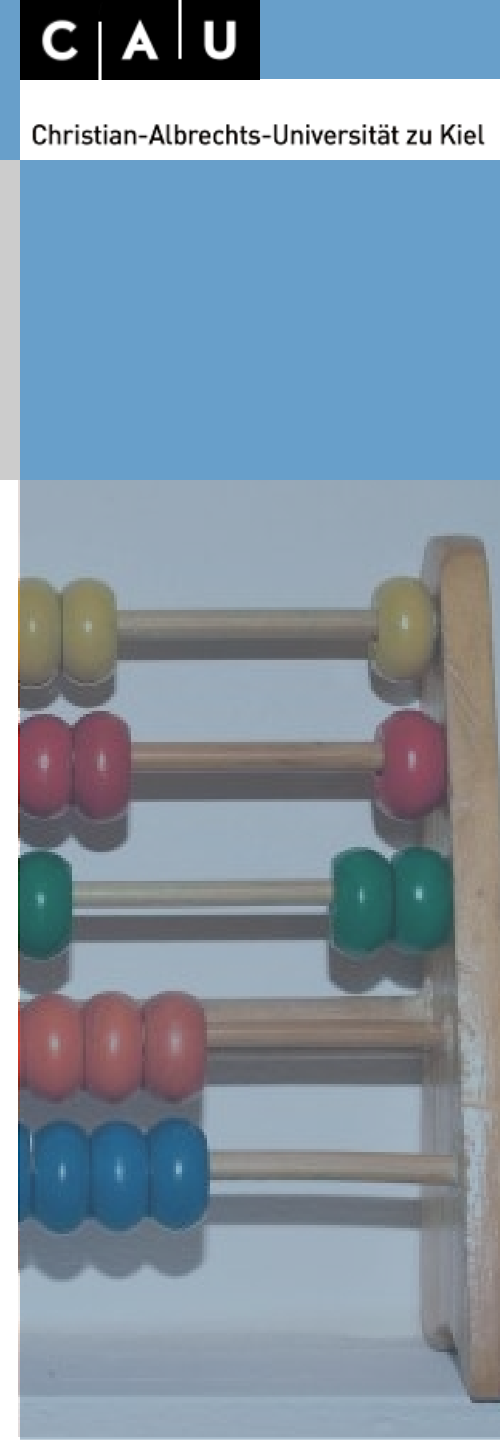
```
> koerpergroessen[koerpergroessen>175]
Hannes      Kay  Björn Martin
  182     198   179   181
> which(koerpergroessen>175)
Hannes      Kay  Björn Martin
    3         6      7       9
> sum(koerpergroessen>175)/length(koerpergroessen)
[1] 0.4444444
```



Faktoren

Dienen zur Kodierung von nominalen Werten:

```
> geschlecht <- factor(c("w", "w", "m", "m", "m", "m",  
"m", "w", "m"))  
> geschlecht  
[1] w w m m m m m w m  
Levels: m w
```



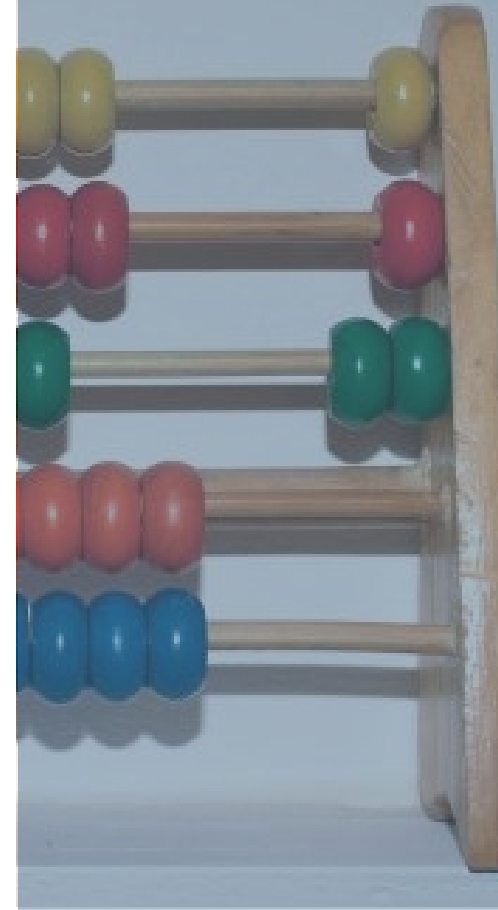
Fehlende (NA) Werte

Problem: Angaben fehlen

```
> alter<-c(26,24,25,23,23,24,30,20,0)
> names(alter)<-kursteilnehmer
> alter
      Ria      Anja      Hannes      Moritz      Basti      Kay      Björn      Cristin      Martin
      26       24       25       23       23       24       30       20       0
> mean(alter)
[1] 21.66667
> sum(alter)/8
[1] 24.375
```

Daher: als N(ot)A(vailable) angeben

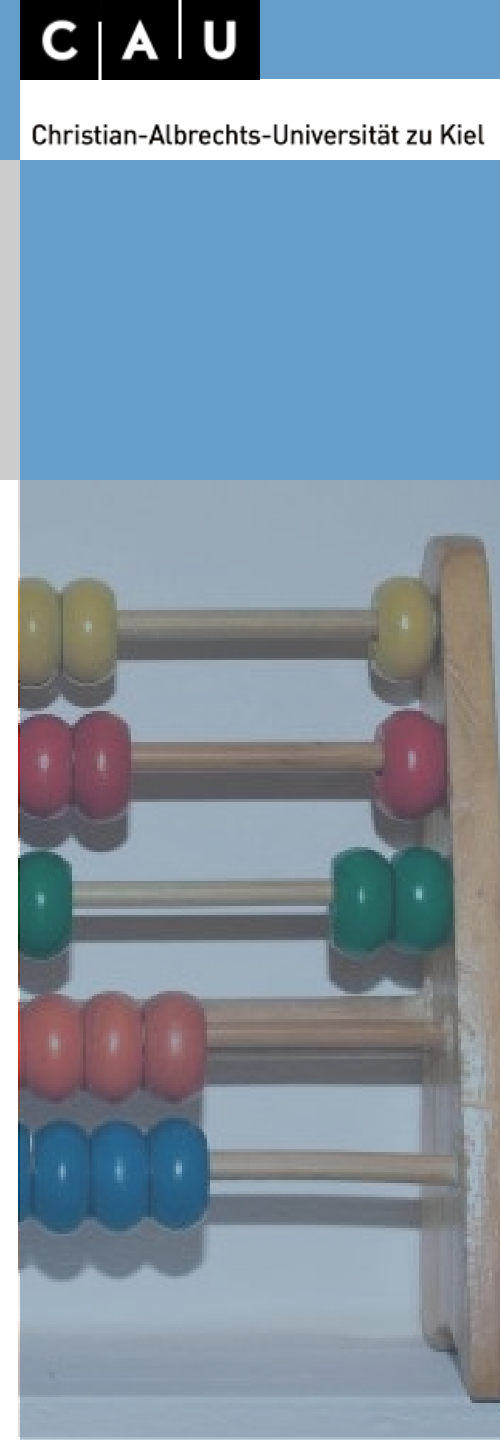
```
> alter<-c(26,24,25,23,23,24,30,20,NA)
> names(alter)<-kursteilnehmer
> alter
> alter
[1] 26 24 25 23 23 24 30 20 NA
> mean(alter)
[1] NA
> mean(alter,na.rm=T)
[1] 24.375
>
```



Matrizen 1

Daten gleicher Art (Zahlen, Faktoren...)

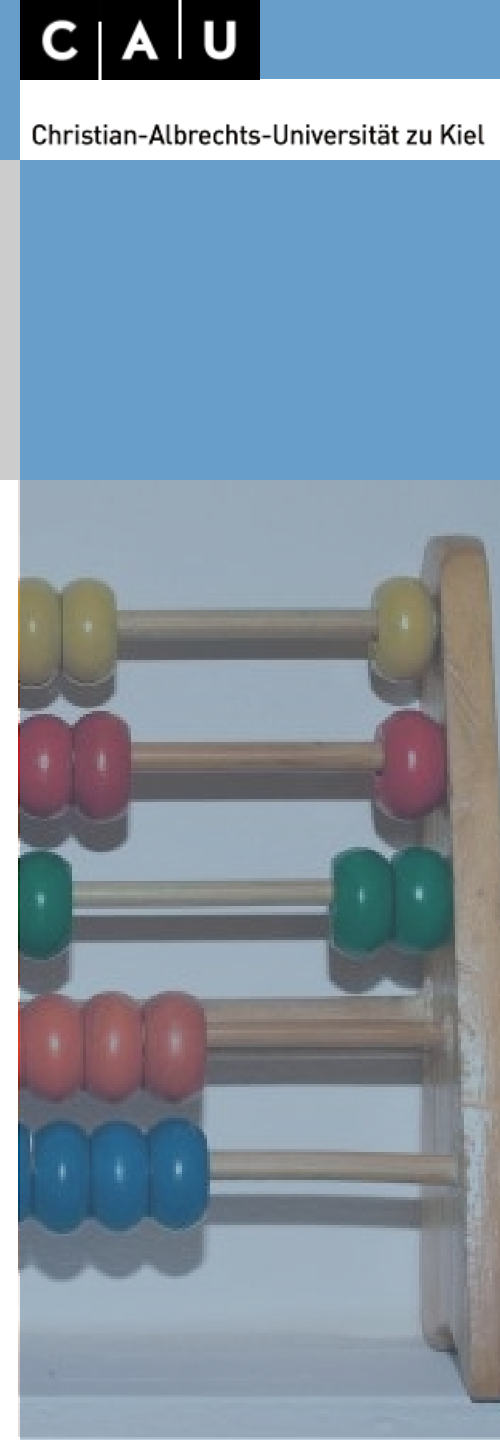
```
> kursmatrix<-matrix(c(koerpergroessen,alter),9,2)
> kursmatrix
      [,1] [,2]
[1,]  174   26
[2,]  163   24
[3,]  182   25
[4,]  175   23
[5,]  173   23
[6,]  198   24
[7,]  179   30
[8,]  163   20
[9,]  181   NA
> rownames(kursmatrix)<-kursteilnehmer
> colnames(kursmatrix)<-c("koerpergroesse","alter")
> kursmatrix
      koerpergroesse alter
Ria                174   26
Anja                163   24
Hannes             182   25
Moritz             175   23
Basti              173   23
Kay                198   24
Björn              179   30
Cristin            163   20
Martin             181   NA
```



Matrizen 2

Funktionen auf Matrizen

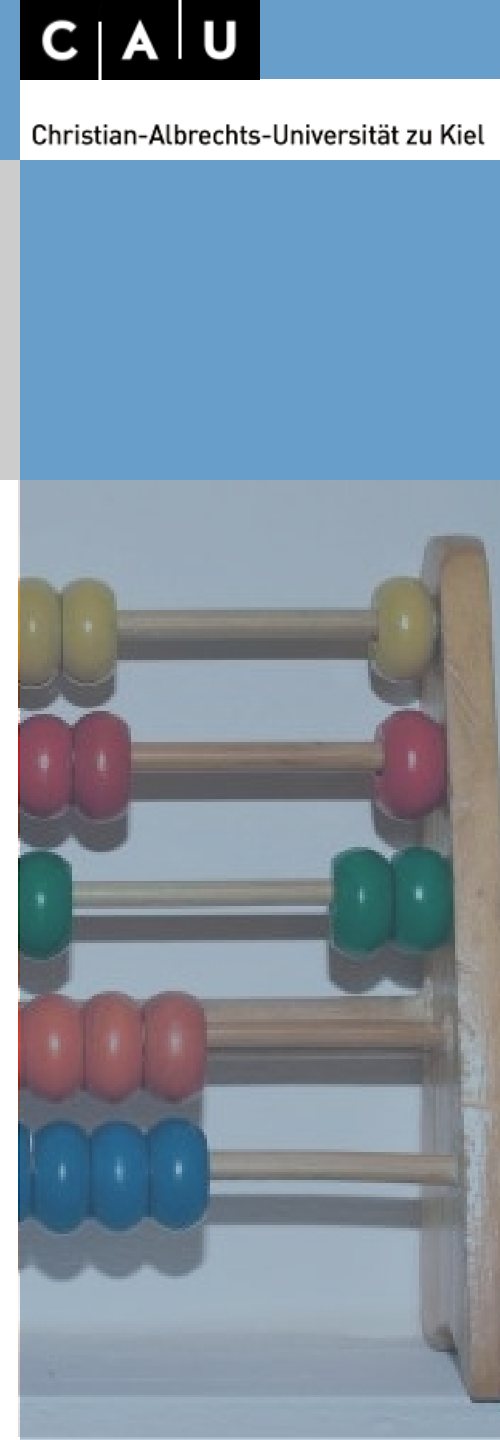
```
> dim(kursmatrix)
[1] 9 2
> length(kursmatrix)
[1] 18
> kursmatrix[3,]
koerpergroesse      alter
          182          25
> kursmatrix[,1]
      Ria      Anja  Hannes  Moritz   Basti      Kay  Björn  Cristin  Martin
      174      163      182      175      173      198      179      163      181
> t(kursmatrix)
      koerpergroesse  Ria  Anja  Hannes  Moritz  Basti  Kay  Björn  Cristin  Martin
alter              26   24    25     23    23   24    30     20     NA
```



Matrizen 3

Funktionen auf Matrizen

```
> kursmatrix/100
      koerpergroesse alter
Ria      1.74    0.26
Anja      1.63    0.24
Hannes     1.82    0.25
Moritz     1.75    0.23
Basti      1.73    0.23
Kay        1.98    0.24
Björn      1.79    0.30
Cristin    1.63    0.20
Martin     1.81     NA
> kursmatrix[,1]/100
      Ria      Anja      Hannes      Moritz      Basti      Kay      Björn      Cristin      Martin
1.74    1.63    1.82    1.75    1.73    1.98    1.79    1.63    1.81
> kursmatrix / c(100,200,300,400,500,1,1,1,1,1,1,1,1,1,1,1,1)
      koerpergroesse alter
Ria      1.7400000    26
Anja      0.8150000    24
Hannes     0.6066667    25
Moritz     0.4375000    23
Basti      0.3460000    23
Kay      198.0000000    24
Björn     179.0000000    30
Cristin    163.0000000    20
Martin     181.0000000     NA
```



Data-Frames

Daten unterschiedlicher Art (Zahlen und Faktoren und ...):

```
> kursdaten<-data.frame(alter,koerpergroessen,geschlecht)
```

```
> kursdaten
```

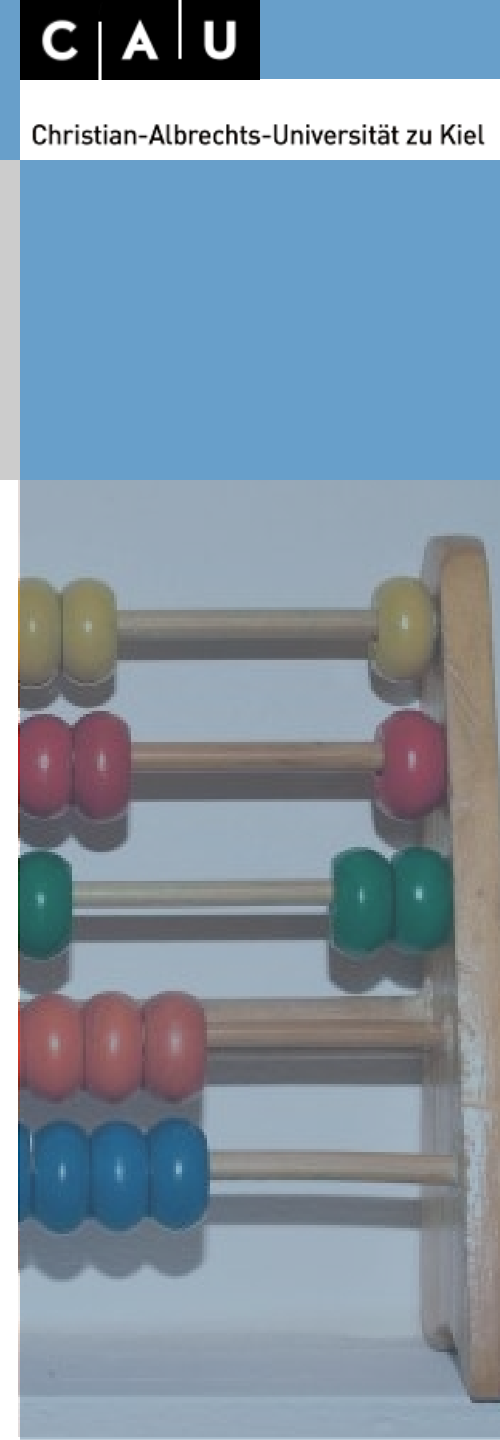
	alter	koerpergroessen	geschlecht
Ria	26	174	w
Anja	24	163	w
Hannes	25	182	m
Moritz	23	175	m
Basti	23	173	m
Kay	24	198	m
Björn	30	179	m
Cristin	20	163	w
Martin	NA	181	m

```
> kursdaten[, "alter"]
```

```
[1] 26 24 25 23 23 24 30 20 NA
```

```
> kursdaten$alter
```

```
[1] 26 24 25 23 23 24 30 20 NA
```



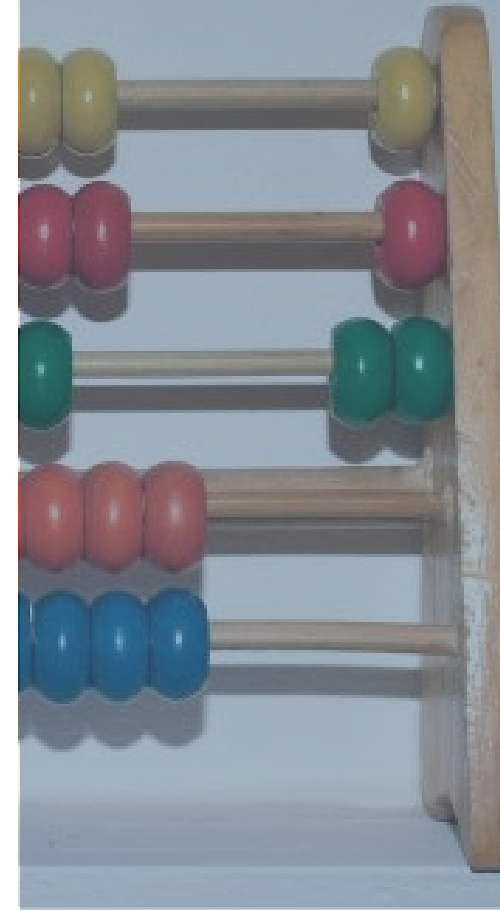
Data-Frames

Funktionen auf Data-Frames

```
> kursdaten$koerpergroessen/100
[1] 1.74 1.63 1.82 1.75 1.73 1.98 1.79 1.63 1.81

> summary(kursdaten)
   alter      koerpergroessen geschlecht
Min.   :20.00   Min.   :163.0    m:6
1st Qu.:23.00   1st Qu.:173.0    w:3
Median :24.00   Median :175.0
Mean   :24.38   Mean   :176.4
3rd Qu.:25.25   3rd Qu.:181.0
Max.   :30.00   Max.   :198.0
NA's   : 1.00

> tapply(kursdaten$alter, kursdaten$geschlecht, mean, na.rm="T")
      m      w
25.00000 23.33333
```



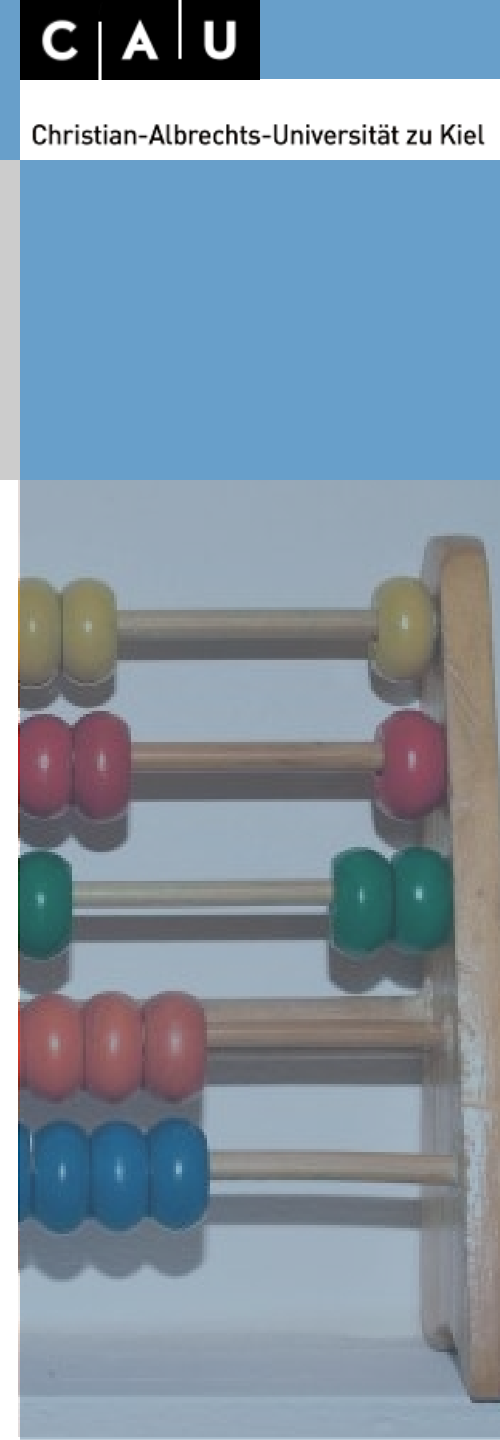
Eingebaute Datensets

Test-Daten zum experimentieren:

```
> data()
```

Data sets in package 'datasets':

AirPassengers	Monthly Airline Passenger Numbers 1949–1960
BJsales	Sales Data with Leading Indicator
BJsales.lead (BJsales)	Sales Data with Leading Indicator
BOD	Biochemical Oxygen Demand
CO2	Carbon Dioxide uptake in grass plants
ChickWeight	Weight versus age of chicks on different diets
DNase	Elisa assay of DNase
EuStockMarkets	Daily Closing Prices of Major European Stock Indices, 1991–1998
Formaldehyde	Determination of Formaldehyde
HairEyeColor	Hair and Eye Color of Statistics Students
Harman23.cor	Harman Example 2.3
Harman74.cor	Harman Example 7.4
Indometh	Pharmacokinetics of Indomethicin
...	

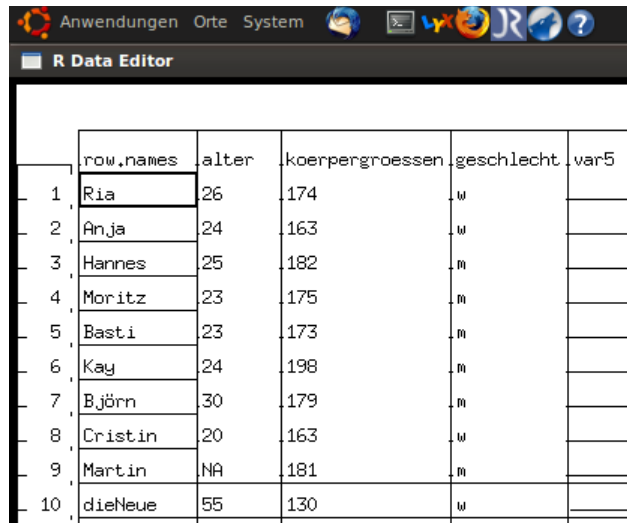


Daten ändern, der bequemere Weg...

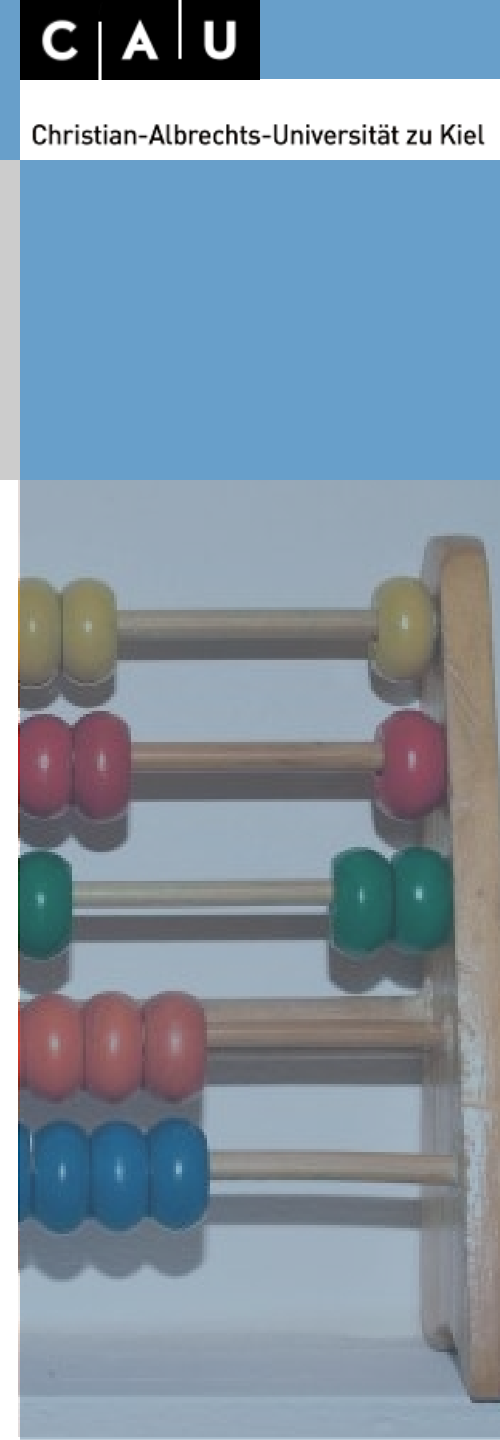
Zum Ändern bereits eingegebener Datensätze:

```
>kursdaten.neu<-edit(kursdaten)  
> kursdaten.neu
```

	alter	koerpergroessen	geschlecht
Ria	26	174	w
Anja	24	163	w
Hannes	25	182	m
Moritz	23	175	m
Basti	23	173	m
Kay	24	198	m
Björn	30	179	m
Cristin	20	163	w
Martin	NA	181	m
dieNeue	55	130	w



	row.names	alter	koerpergroessen	geschlecht	var5
1	Ria	26	174	w	
2	Anja	24	163	w	
3	Hannes	25	182	m	
4	Moritz	23	175	m	
5	Basti	23	173	m	
6	Kay	24	198	m	
7	Björn	30	179	m	
8	Cristin	20	163	w	
9	Martin	NA	181	m	
10	dieNeue	55	130	w	



Datenausgabe durch Schreiben von Dateien

Einfache Textdatei:

```
> write(kursmatrix,"kursmatrix.txt")
```

Data-Frame als einfache Textdatei:

```
> write.table(kursdaten,"kursdaten.txt")
```

Data-Frame als csv-Datei:

```
> write.csv2(kursdaten,"kursdaten.csv")
```

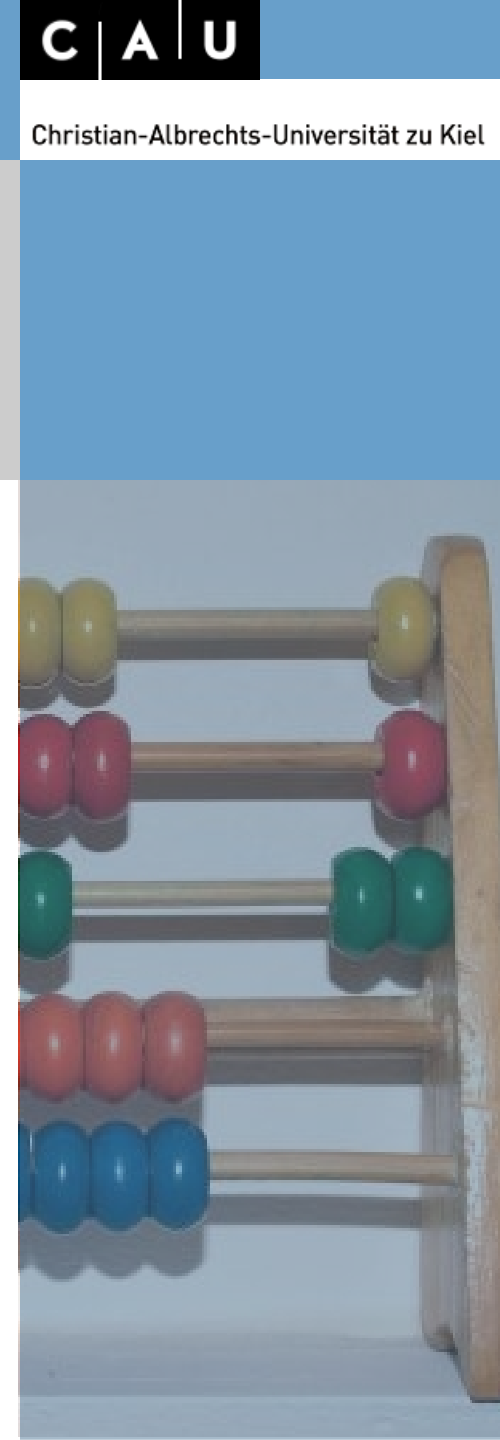
Achtung: Dezimalzeichen ist . nicht ,

```
> kursdaten$koerpergroessen<-kursdaten$koerpergroessen/100  
> write.csv2(kursdaten,"kursdaten.csv")
```

-Probleme beim Einlesen in z.B. Excel-

daher:

```
> write.csv2(kursdaten,"kursdaten.csv",dec=",")  
Warning message:  
In write.csv2(kursdaten, "kursdaten.csv", dec = ",") :  
  Versuch 'dec' auf ignoriert zu setzen  
>
```



Dateneingabe durch Einlesen von Dateien

Zur Erinnerung:

```
> getwd()  
[1] "/home/martin" # oder etwas anderes...  
> setwd("U:\R") # oder etwas anderes...
```

Einfache Textdatei:

```
> kursmatrix.gelesen<-matrix(scan("kursmatrix.txt"),ncol=2)
```

Data-Frame als einfache Textdatei:

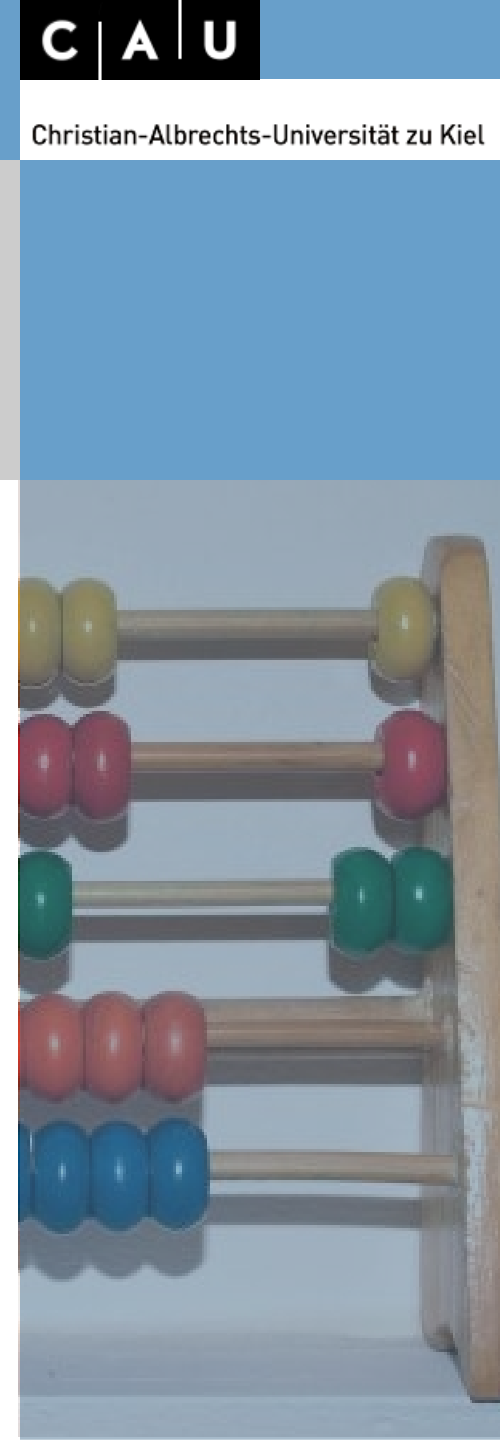
```
> kursdaten.gelesen<-read.table("kursdaten.txt")
```

Data-Frame als csv-Datei:

```
> kursdaten.gelesen<-read.csv2("kursdaten.csv")
```

Lesen mit Zeilennamen

```
> kursdaten.gelesen<-read.csv2("kursdaten.csv",row.names=1)
```

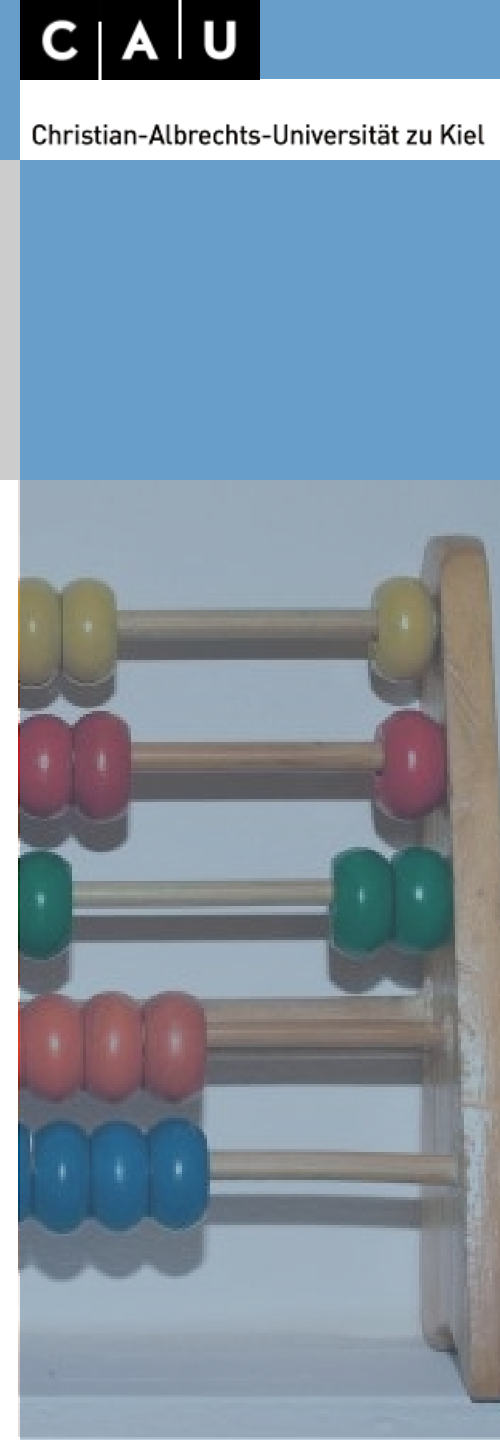


Grundlegende statistische Verfahren für archäologische Datenanalyse in R

R <-> Excel

Speichern immer als CSV

Es gibt zwar Pakete für R, die Excel-Daten lesen und schreiben können, für diese sind allerdings weitere Installationen nötig (Perl, Python u.a.)



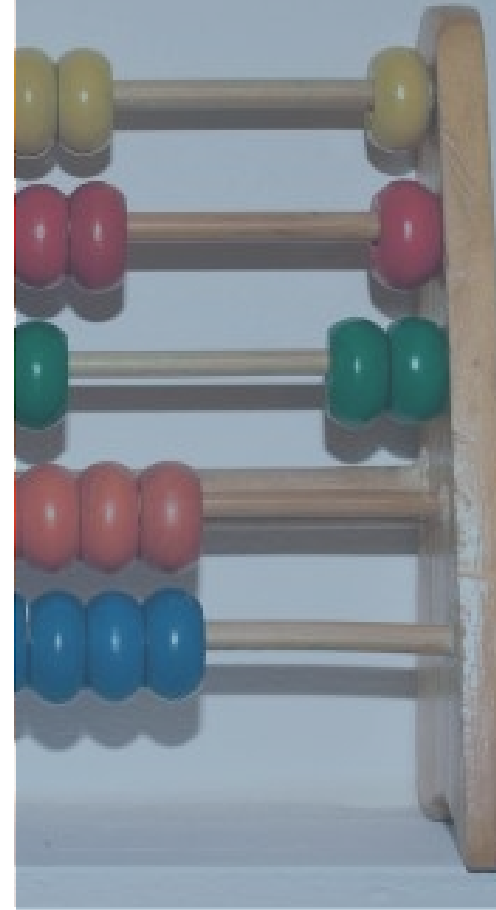
R beenden


`q()`

Save workspace image? [y/n/c]:

Speichert die aktuelle Umgebung mit allen Variablen für die nächste Session ab, wenn gewünscht

y	speichern
n	nicht speichern
c	nicht beenden





Nächste Sitzung 11. November:
Explorative Statistik
(graphische Darstellung)

Weiterführendes Material unter
<http://cran.r-project.org/>
Besonders „Manuals“