ETH zürich

Züchtungslehre - Einführung in R

Peter von Rohr

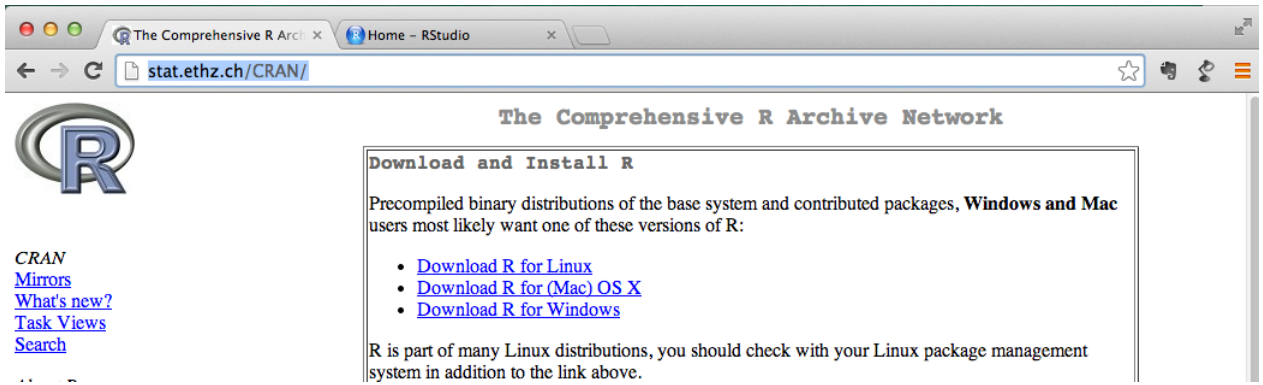
Qualitas AG

Einführung in R - Wieso R

- Interpretierte Sprache, d.h. ein Programm - der sogenannte Interpreter - wartet auf Eingaben
- Einfache Installation: Download und Installation, analog zu anderen Programmen
- Verfügbarkeit: Windows, Mac, Linux
- Gratis und offen, d.h. Programm-code ist verfügbar
- Gute Unterstützung durch externe Tools, z. Bsp RStudio
- Grosse Benutzergemeinde, da sehr populär in Statistik
- Flexibles Arbeiten und schnelles Prototyping, d.h. Ideen sind sehr schnell in Programmcode verwandelt
- Gute Grafikfähigkeiten, d.h. Daten können schnell und einfach visualisiert werden

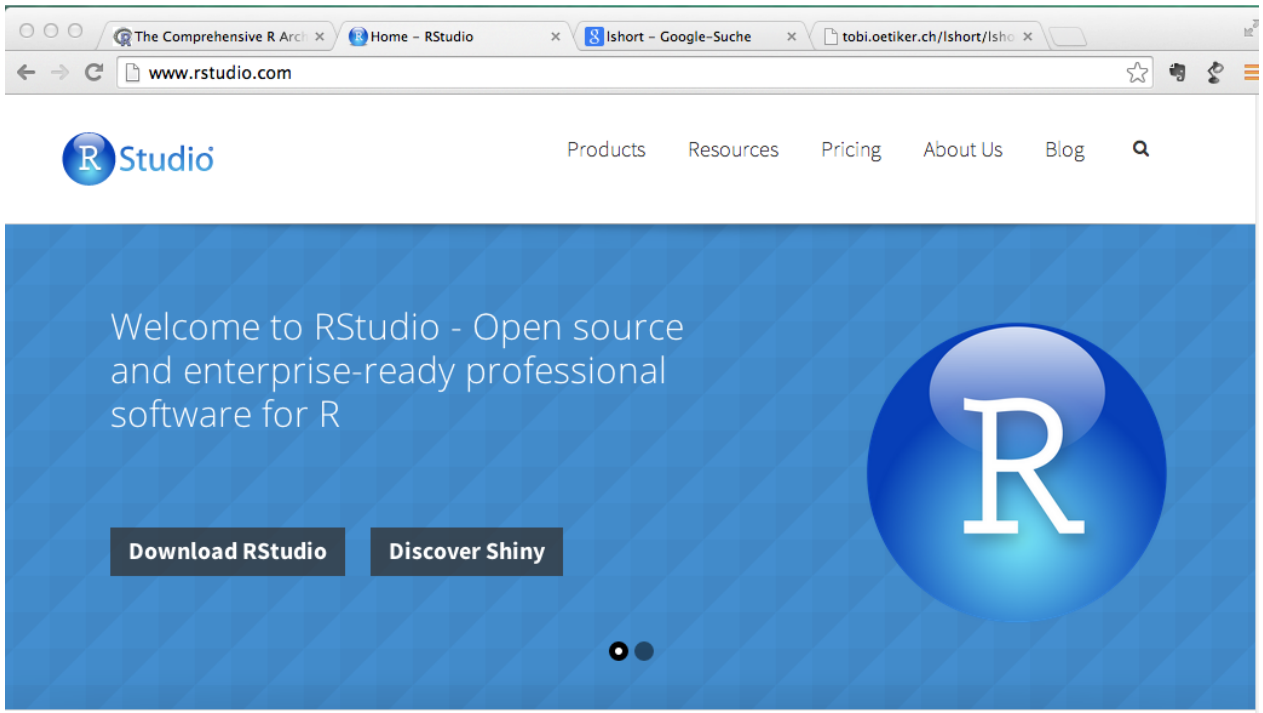
Installation

- Download von CRAN (Comprehensive R Archive Network): http://stat.ethz.ch/CRAN/
- Installation des Binaries (.exe unter Windows und .pkg unter Mac) mit Doppelclick auf Datei
- Webseite von R: https://www.r-project.org/



Installation von RStudio

- Download von https://www.rstudio.com
- Installation des Binaries (verlangt, dass R schon installiert ist)



Erste Sitzung mit R - Interaktiver Modus

- R kann wie ein Taschenrechner verwendet werden $\hat{=}$ *interaktiver Modus*

```
> 5 + 7

[1] 12

> 9*3

[1] 27
```

- Es gilt Klammer vor Punkt vor Strich

```
> 45 - 15 / 2

[1] 37.5

> (45 - 15) / 2

[1] 15
```

Erste Sitzung mit R - Mehr Operatoren

- Quadratwurzel

```
> sqrt(2)
```

```
[1] 1.414214
```

- Potenz

```
> 4.92 ^ 3
```

```
[1] 119.0955
```

- Logarithmus

```
> log(15)
```

```
[1] 2.70805
```

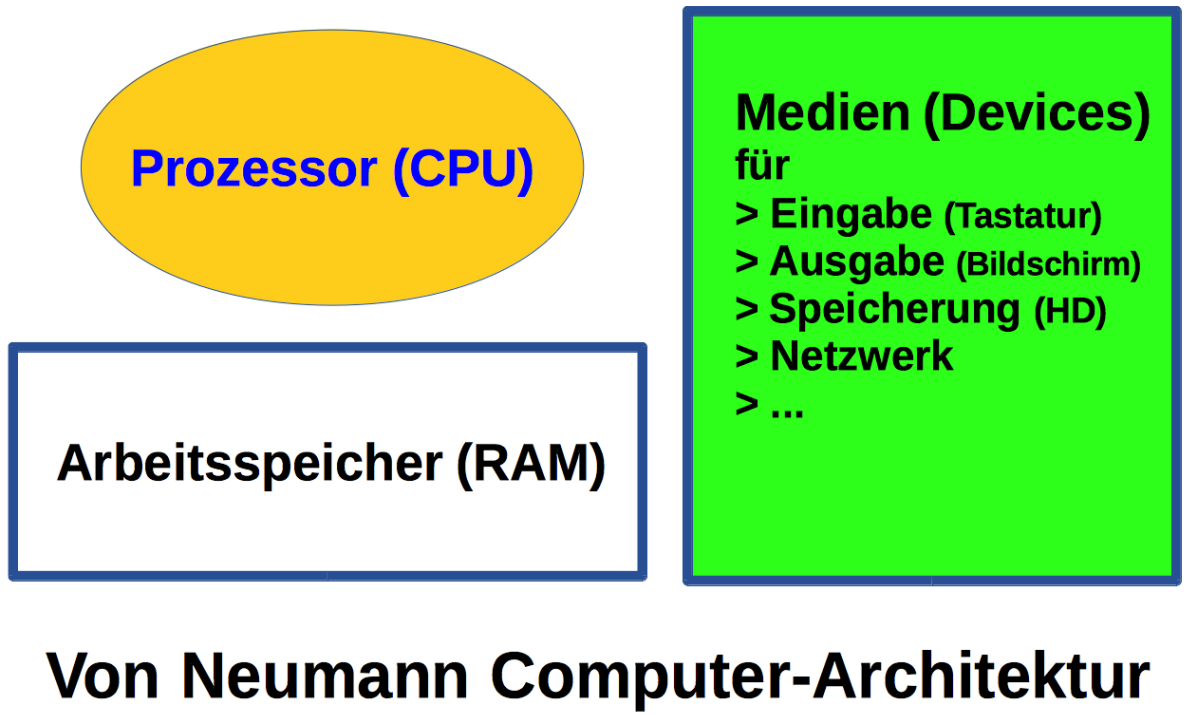
Variablen = Objekte

- Interaktiver Modus mühsam, falls längere Berechnungen mit Zwischenergebnissen
- Beispiel: Mittelwert m und Standardabweichung s von fünf Zahlen
- Mittelwert: Interaktiv

```
> (15 + 9 + 8 + 34 + 76) / 5

[1] 28.4

> sqrt(((15-28.4)^2 + (9-28.4)^2 + (8-28.4)^2
+         + (34-28.4)^2 + (76-28.4)^2)/4)

[1] 28.58846
```

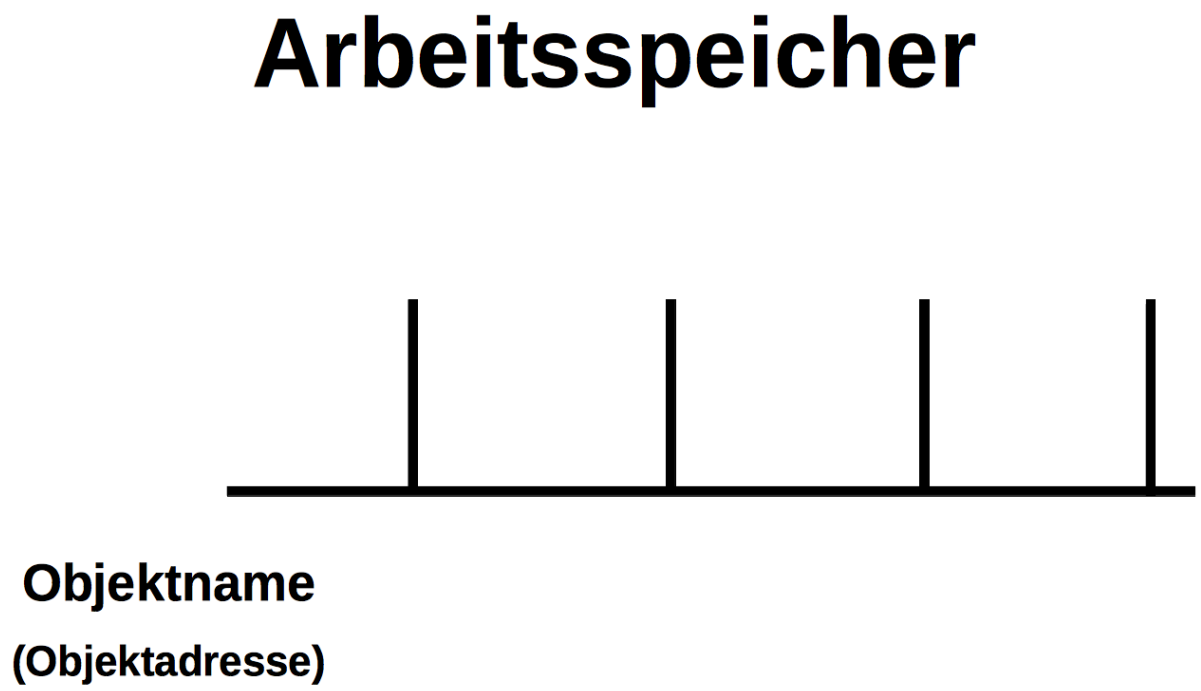
Variablen = Objekte II

- Verbesserung: Ablage von Zwischenresultaten im Arbeitsspeicher
- Wie sieht das aus im Arbeitsspeicher eines Computers?



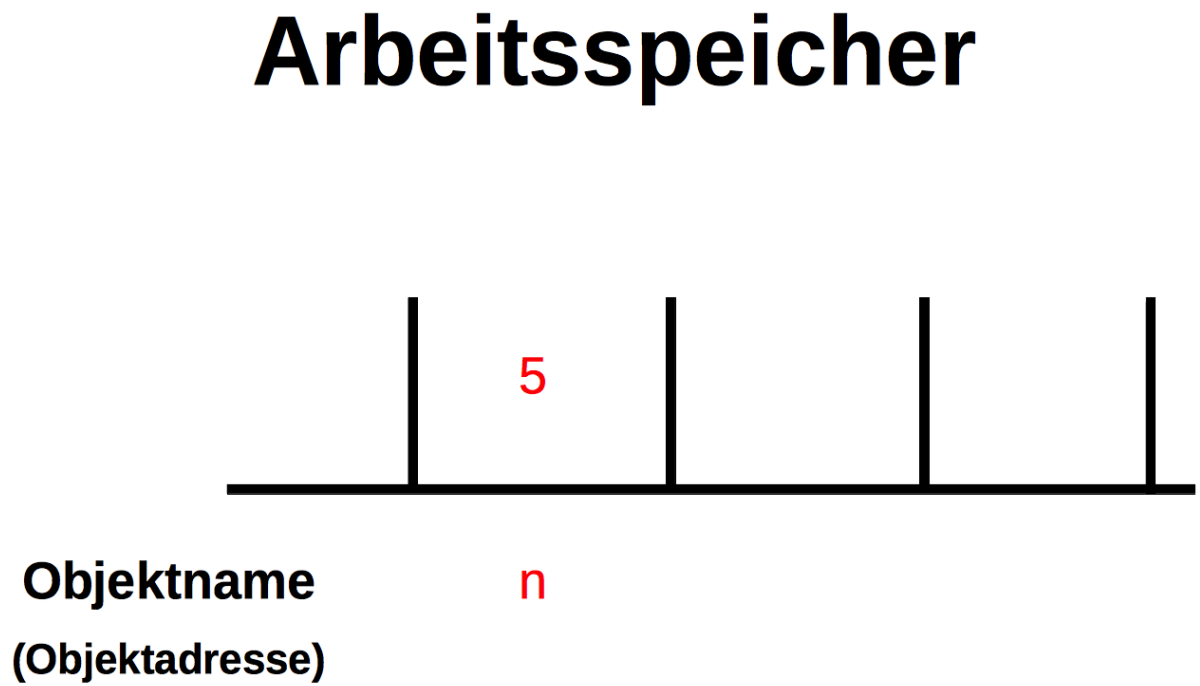
Von Neumann Computer-Architektur

Arbeitsspeicher



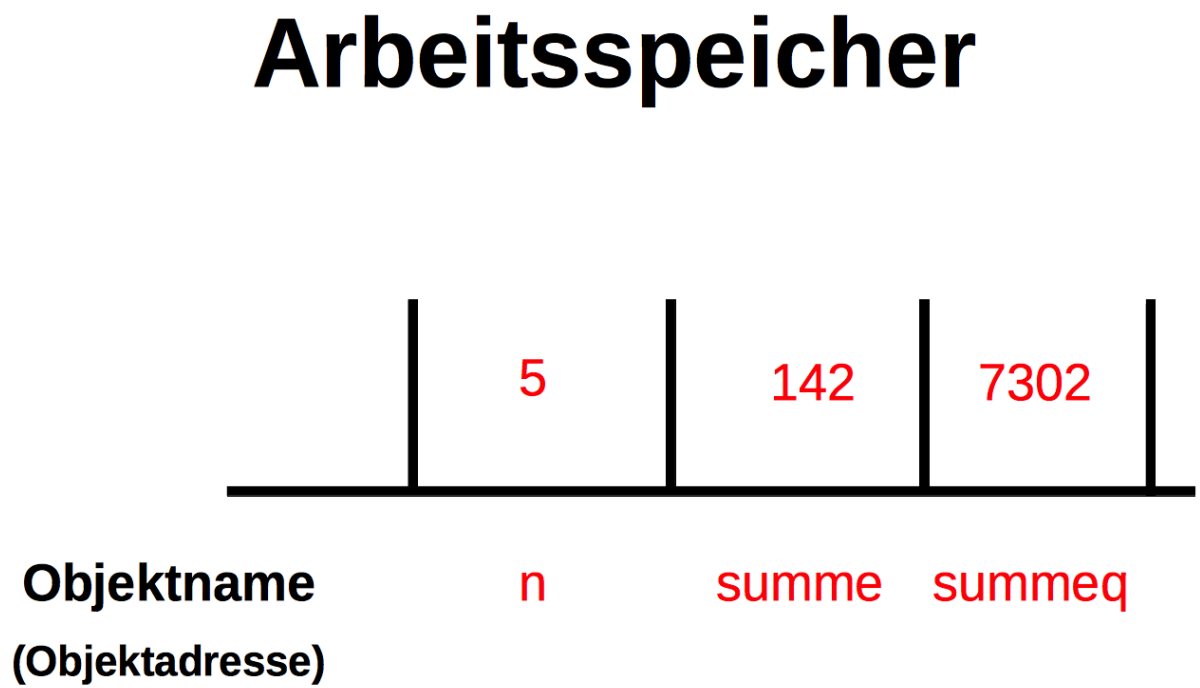
Zuweisung von Werten zu Objekten

```
> n <- 5
```



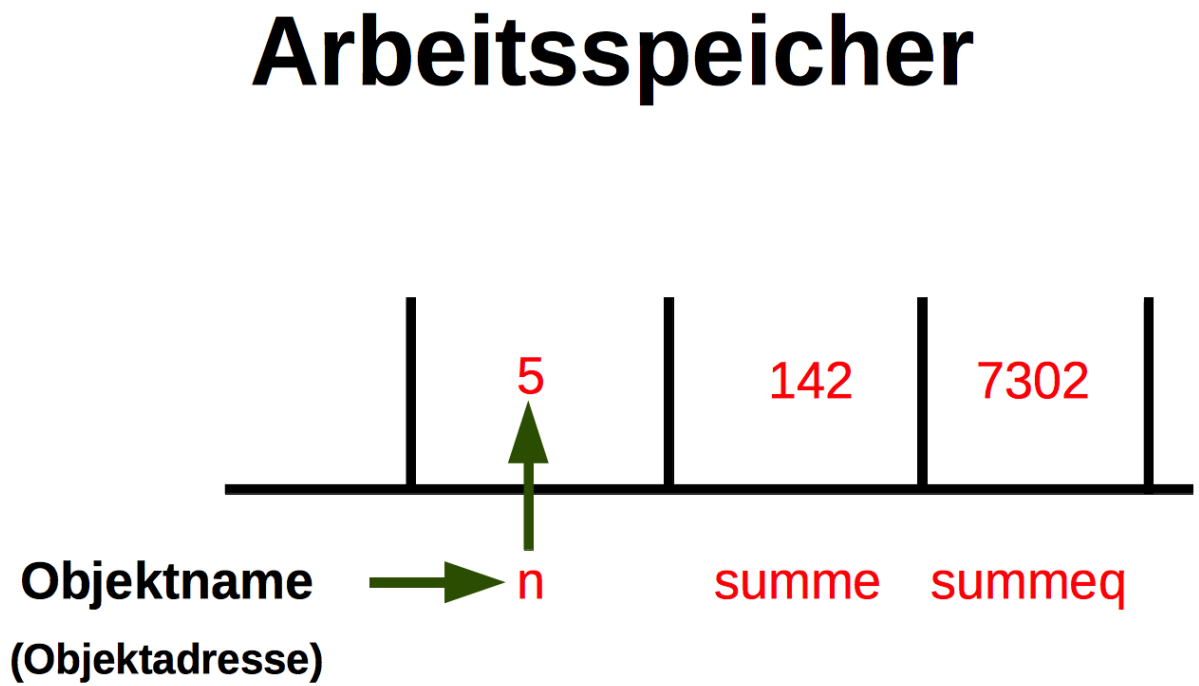
Zuweisung von Werten zu Objekten II

```
> summe <- 15 + 9 + 8 + 34 + 76
> summeq <- 15^2 + 9^2 + 8^2 + 34^2 + 76^2
```



Zugriff auf Werte

```
> n
[1] 5
```



Namen von Objekten

- Kann Buchstaben, Zahlen und Zeichen wie "-", "_" oder "." enthalten
- Nicht mit einer Zahl beginnen
- Reservierte Namen wie c, i, t, oder andere Funktionsnamen nicht als Objektnamen verwenden
- Allgemeine Hinweise zu Style, Namen und Notationen: http://r-pkgs.had.co.nz/style.html

Rechnen mit Objekten

- Mittelwert m

```
> m <- summe / n
```

- Standardabweichung s

```
> s <- sqrt((summeq - summe^2/n)/(n-1))
```

- Der R-interpreter gibt bei Zuweisungen kein Resultat zurueck (ausser man macht eine Klammer)
- Output wird angezeigt, wenn Objektname eingegeben wird oder mit der `print()`-Funktion

```
> m
```

```
[1] 28.4
```

```
> print(s)
```

```
[1] 28.58846
```

Erzeugung von Objekten mit Funktionen

- Bisher: Objekte durch Zuweisungen erzeugt
- Funktionen `sqrt()` und `print()` schon angetroffen
- Beispiel - Vektor

```
> (vecNum <- vector(mode = "numeric", length = 2))
[1] 0 0
```

- Hilfe zu einer Funktion mit

```
> help("vector")
> ?vector
```

Datentypen in R

Es werden sechs Datentypen unterschieden

1. `numeric`: reelle Zahlen, default für Zahlen in R
2. `integer`: ganze Zahlen (0, 1, 2, ...) mit "L" hinter der Zahl
3. `complex`: Quadratwurzeln aus negativen Zahlen
4. `character`: Buchstaben, Zeichen, ...
5. `factor`: spezieller Datentype für lineare Modelles

Wichtige Punkte zu Datentypen

Datentypen - Prüfung und Umwandlung

- R kennt keine strenge Prüfung von Datentypen (strong typing)
- R konvertiert Datentypen automatisch (coersion)
- Coersion ist bequem, kann aber auch Fehlerquelle sein

Hilfreiche Funktionen

- `class()` gibt den Type eines Objekts zurück
- `is.<data.type>()` prüft, ob ein Objekt von gewissen Datentyp ist, z. Bsp `is.integer(5)`
- `as.<data.type>()` kann für eine explizite Umwandlung verwendet werden z. Bsp `as.character(12)`

Vektoren

- Ein Vektor ist eine Sequenz von Datenelementen vom gleichen Basistyp
- Datenelemente in einem Vektor werden als Komponenten bezeichnet
- In R werden Vektoren mit der Funktion `vector()` erzeugt und mit Funktion `c()` erweitert.
- Beispiel:

```
> vecNum <- vector(mode = "numeric", length = 2)
> vecNum[1] <- 5
> vecNum[2] <- -1
> print(vecNum)
[1]  5 -1
```

Vektoren II

- ... oder in einem Schritt mit `c()`

```
> vecNum <- c(5,-1)
> print(vecNum)
```

```
[1]  5 -1
```

- Erweiterung bestehender Vektoren mit

```
> vecChar <- c("aa","bb","da")
> vecChar <- c(vecNum, vecChar)
> print(vecChar)
```

```
[1] "5"  "-1" "aa" "bb" "da"
```

Eigenschaften von Vektoren

- Anzahl Komponenten in einem Vektor

```
> length(vecChar)

[1] 5
```

- Test ob ein Vektor leer ist, also keine Elemente enthält

```
> length(vecChar) == 0

[1] FALSE

> vecLogical <- vector(mode = "logical", length = 0)
> length(vecLogical) == 0

[1] TRUE
```

Logische Operationen mit Vektoren

- Logische oder Boolsche Operationen (grösser, kleiner, gleich) auf Vektoren werden elementweise ausgeführt
- Resultat entspricht einem Vektor der gleichen Länge wie der ursprüngliche Vektor mit TRUE und FALSE Werten
- Beispiel:

```
> a <- c(33,5,7,13,-1)
> a > 5
```

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

Vektorarithmetik - Plus, Minus

- Arithmetische Operationen mit Vektoren (plus, minus, mal, durch) werden elementweise ausgeführt
- Beispiel

```
> x <- c(3,5,13,-2)
> y <- c(2,6,-3,19)
> x+y
[1]  5 11 10 17
> x-y
[1]   1  -1  16 -21
```

Vektorarithmetik - Mal, Durch

- Elementweise Ausführung analog zu Plus, Minus

```
> x*y
[1]   6  30 -39 -38
> x/y
[1]  1.5000000  0.8333333 -4.3333333 -0.1052632
```

- Skalarprodukt zwischen zwei Vektoren mit

```
> crossprod(x,y)
     [,1]
[1,]  -41
```

Zugriff auf Komponenten im Vektor

- Numerischer Index

```
> x[2]

[1] 5
```

- Negativem Indices schliessen Komponenten aus

```
> x[-3]

[1]  3  5 -2
```

- Bereiche von Indices

```
> x[3:4]

[1] 13 -2
```

Matrix

- Matrix = Stapel von mehreren Vektoren gleicher Länge, d.h. Matrizen sind zweidimensionale Objekte, welche in Kolonnen und Spalten organisiert sind
- Komponenten von Matrizen sind vom gleichen Typ
- Beispiel:

```
> matA <- matrix(c(5,3,4,-6,3,76), nrow = 2, ncol = 3,
+                byrow = TRUE)
> print(matA)
     [,1] [,2] [,3]
[1,]    5    3    4
[2,]   -6    3   76
```

Dimension einer Matrix

- Eine grundlegende Eigenschaft einer Matrix ist ihre Dimension
- Dimension = Anzahl Zeilen und Anzahl Kolonnen
- Darstellung als Vektor der Länge zwei
- Beispiel

```
> dim(matA)
[1] 2 3
```

Zugriff auf Matrix Elemente

- Per numerischem Index

```
> matA[2,1]
```

```
[1] -6
```

- Zugriff auf ganze Zeile

```
> matA[1,]
```

```
[1] 5 3 4
```

- Zugriff auf ganze Spalte

```
> matA[,2]
```

```
[1] 3 3
```

Zeilenweise Erweiterung einer Matrix

```
> matB <- matrix(c(3,-1,90,1,1,4), nrow = 2, ncol = 3,
+                byrow = TRUE)
> rbind(matA, matB)
     [,1] [,2] [,3]
[1,]    5    3    4
[2,]   -6    3   76
[3,]    3   -1   90
[4,]    1    1    4
```

Kolonnenweise Erweiterung einer Matrix

```
> cbind(matA, matB)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    5    3    4    3   -1   90
[2,]   -6    3   76    1    1    4
```

Spezielle Matrizen: Transponierte

- Transponierte: Zeilen werden Kolonnen und Kolonnen werden Zeilen

```
> t(matA)
```

```
     [,1] [,2]
[1,]    5   -6
[2,]    3    3
[3,]    4   76
```

- Transponierte der Transponierten gibt ursprüngliche Matrix

```
> t(t(matA))
```

```
     [,1] [,2] [,3]
[1,]    5    3    4
[2,]   -6    3   76
```

Spezielle Matrizen: Nullmatrix

- Alle Elemente sind gleich 0

```
> mat0 <- matrix(data=rep(0,6), nrow=2, ncol = 3)
> print(mat0)
```

```
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
```

Spezielle Matrizen: Einheitsmatrix

- Diagonalelemente gleich 1, alle anderen gleich 0

```
> matI <- diag(1, nrow = 3, ncol = 3)
> print(matI)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

Spezielle Matrizen: Inverse

- Matrix mal ihre Inverse ergibt Einheitsmatrix

```
> matD <- matrix(data = c(15,3,2,27), nrow = 2,ncol = 2)
> (matDInv <- solve(matD))
```

```
              [,1]          [,2]
[1,]  0.067669173 -0.005012531
[2,] -0.007518797  0.037593985
```

Addition und Subtraktion

- Addition und Subtraktion werden element-weise ausgeführt

```
> matA + matB
```

```
     [,1] [,2] [,3]
[1,]    8    2   94
[2,]   -5    4   80
```

```
> matA - matB
```

```
     [,1] [,2] [,3]
[1,]    2    4  -86
[2,]   -7    2   72
```

Matrix Multiplikation

- Regel: Zeile mal Kolonne
- Dimensionen müssen stimmen

```
> matA %*% t(matB)
```

```
     [,1] [,2]
[1,]  372   24
[2,] 6819  301
```

- Kontrolle der inversion Matrix

```
> matDInv %*% matD
```

```
     [,1]          [,2]
[1,]    1 2.775558e-17
[2,]    0 1.000000e+00
```

Liste

- Eine Liste fasst eine Sammlung von Objekten zusammen.
- Einzelne Objekte heissen *Komponenten*
- Komponenten innerhalb der Liste haben eine fixe Reihenfolge
- Komponenten müssen nicht vom selben Typ sein
- Beispiel

```
> lstA <- list(nZahlen = c(5,-2,7),
+              sNamen = c("Fred","Mary"),
+              lBool = c(FALSE,TRUE))
```

Zugriff auf Listenelemente

- Mit Index und einfacher Klammerung, Resultat ist wieder eine Liste

```
> lstA[2]

$sNamen
[1] "Fred" "Mary"
```

- Mit Komponentennamen

```
> lstA["nZahlen"]

$nZahlen
[1]  5 -2  7
```

Zugriff auf Listenelemente II

- Mit Index und doppelter Klammer, Resultat ist Vektor

```
> lstA[[1]]

[1]  5 -2  7
```

- Mit $ und Komponentennamen

```
> lstA$lBool

[1] FALSE  TRUE
```

Spezielle Listen - Dataframes

- Ein *Dataframe* ist eine Liste von Vektoren, welche alle die gleiche Länge haben
- Erzeugung mit Funktion `data.frame()`

```
> dfA <- data.frame(nZahl = c(-2,15),
+                   sZeichen = c("Alice","Bob"),
+                   bWahr = c(FALSE,FALSE))
```

- Zugriff, wie bei Matrix mit Indices oder mit Namen

```
> dfA[2,1]

[1] 15

> dfA$nZahl

[1] -2 15
```

Daten in R einlesen

- Eine der wichtigsten Aufgaben ist, dass externe Daten in R eingelesen werden können
- Die wichtigste Funktion für das Einlesen von Daten: `read.table()` oder darauf aufbauende Funktionen, wie z. Bsp `read.csv2()` zum Einlesen von Daten im CSV-Format
- Häufige Prozedur für den Transfer von Daten aus Excel:
  1. Speichern als Text(csv)
  2. Einlesen in R mit `read.csv2()`

Daten in R einlesen II

- Messungen von Brustumfang und Gewicht als Excel-Tabelle
- Einlesen dieser Daten in R

```
> dfBrGew <- read.csv2(file = "br_gew.csv",
+                      stringsAsFactors = FALSE)
> dim(dfBrGew)
[1] 10  2
> head(dfBrGew, 3)
  Brustumfang Gewicht
1         176     471
2         177     463
3         178     481
```

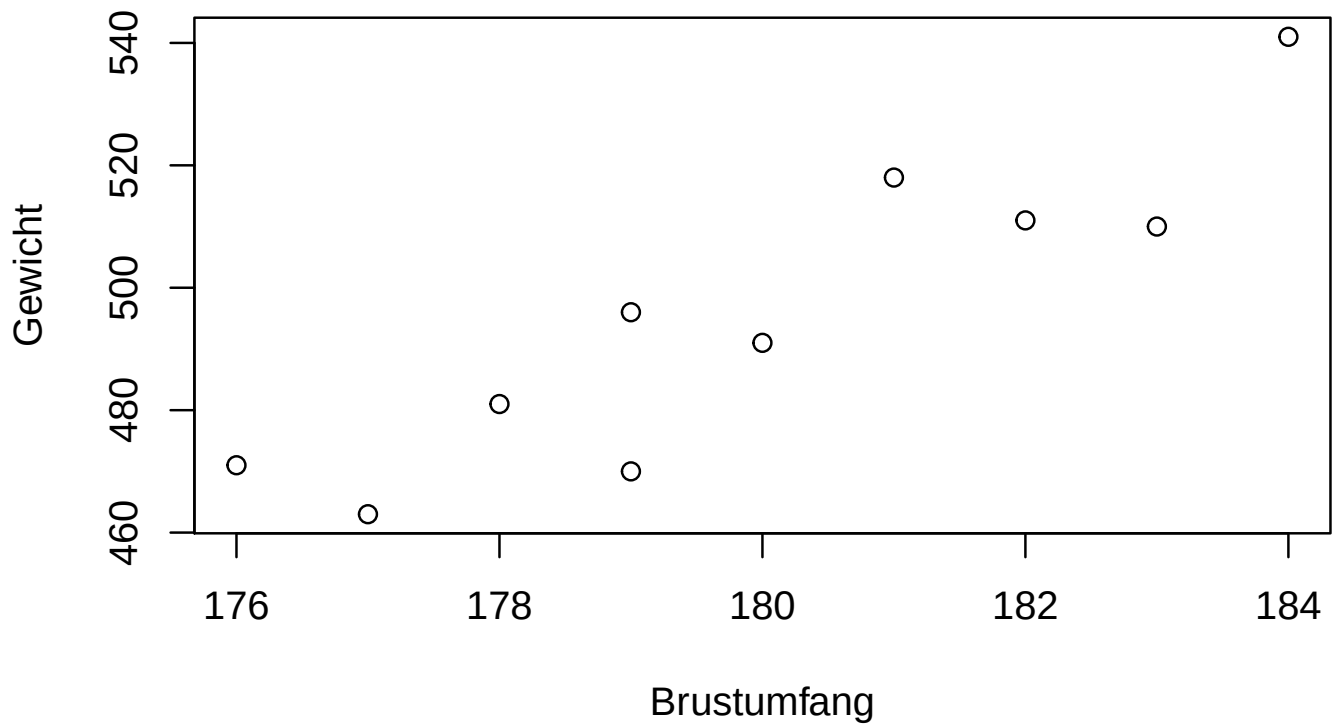
Schreiben von Daten in Dateien

- Dataframes oder Matrizen werden mit der Grundfunktion `write.table()` in Dateien geschrieben
- Analog zu `read.csv2()` gibt es `write.csv2()`
- Für nicht csv-Daten kann `cat()` verwendet werden

Diagramme und Plots

- Plots in R sind einfach: Funktion `plot()`

```
> plot(dfBrGew)
```



Einfaches lineares Modell

- Regressionsmodell von Gewicht auf Brustumfang

```
> lmBrGew <- lm(Gewicht ~ Brustumfang, data = dfBrGew)
> plot(dfBrGew)
> abline(coef = coefficients(lmBrGew), col = "red")
```