Introductionary + advanced R course

Berry Boessenkool, berry-b@gmx.de
Hints and corrections are very welcome!

Download the current slides, source code and datasets at

bookdown.org/brry/course

These slides are licenced under CC BY,
so you can use the material freely as long as you cite me.

PDF created on 2020-12-14, 11:25

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Outline

R course info
1. One-Session-Intro
   1. Intro, objects, vectors
   2. Files, data.frames
   3. Plotting
   4. Packages, regression
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

print("Hello world!")

- Berry Boessenkool → berry-b@gmx.de

print("Hello world!")

- Berry Boessenkool → berry-b@gmx.de
- Geoecology @ Potsdam University

print("Hello world!")

- Berry Boessenkool → berry-b@gmx.de
- Geoecology @ Potsdam University
- R Fan

print("Hello world!")

- Berry Boessenkool → berry-b@gmx.de
- Geoecology @ Potsdam University
- R Fanatic

print("Hello world!")

- Berry Boessenkool → berry-b@gmx.de
- Geoecology @ Potsdam University
- R Fanatic since 2010

`print("Hello world!")`

- Berry Boessenkool → berry-b@gmx.de
- Geoecology @ Potsdam University
- R Fanatic since 2010
    - Programming: `rdwd`,

`print("Hello world!")`

- Berry Boessenkool → berry-b@gmx.de
- Geoecology @ Potsdam University
- R Fanatic since 2010
    - Programming: `rdwd`, `extremeStat`,

`print("Hello world!")`

- Berry Boessenkool → berry-b@gmx.de
- Geoecology @ Potsdam University
- R Fanatic since 2010
  - Programming: `rdwd`, `extremeStat`, `OSMscale`

print("Hello world!")

- Berry Boessenkool → berry-b@gmx.de
- Geoecology @ Potsdam University
- R Fanatic since 2010
  - Programming: `rdwd`, `extremeStat`, `OSMscale`
  - Community: BERLIN R USER GROUP

print("Hello world!")

- Berry Boessenkool → berry-b@gmx.de
- Geoecology @ Potsdam University
- R Fanatic since 2010
  - Programming: `rdwd`, `extremeStat`, `OSMscale`
  - Community: BERLIN R USER GROUP
  - Training & Consulting:

print("Hello world!")

- Berry Boessenkool → berry-b@gmx.de
- Geoecology @ Potsdam University
- R Fanatic since 2010
  - Programming: `rdwd`, `extremeStat`, `OSMscale`
  - Community: BERLIN R USER GROUP
  - Training & Consulting: B
- These slides were originally based on a one week course held in 2013 for CaWa with Mathias Seibert (GFZ) in Bishkek, Kyrgyzstan

print("Hello world!")

- Berry Boessenkool → berry-b@gmx.de
- Geoecology @ Potsdam University
- R Fanatic since 2010
  - Programming: `rdwd`, `extremeStat`, `OSMscale`
  - Community: BERLIN R USER GROUP
  - Training & Consulting:
- These slides were originally based on a one week course held in 2013 for CaWa with Mathias Seibert (GFZ) in Bishkek, Kyrgyzstan
- If we're proceeding too fast, please interrupt!

Session 1: Why R, read file, select and plot data

Aim:

Session 1: Why R, read file, select and plot data

Aim: get a general impression on how R works, without going too much into technical detail.

Session 1: Why R, read file, select and plot data

Aim: get a general impression on how R works, without going too much into technical detail.
Afterwards, you should be able to:

Session 1: Why R, read file, select and plot data

Aim: get a general impression on how R works, without going too much into technical detail.
Afterwards, you should be able to:

- start a new script in Rstudio and execute lines of code

Session 1: Why R, read file, select and plot data

Aim: get a general impression on how R works, without going too much into technical detail.
Afterwards, you should be able to:

- start a new script in Rstudio and execute lines of code
- read a data file into R

Session 1: Why R, read file, select and plot data

Aim: get a general impression on how R works, without going too much into technical detail.
Afterwards, you should be able to:

- start a new script in Rstudio and execute lines of code
- read a data file into R
- select a column and rows that fullfill a certain criterion

Session 1: Why R, read file, select and plot data

Aim: get a general impression on how R works, without going too much into technical detail.
Afterwards, you should be able to:

- start a new script in Rstudio and execute lines of code
- read a data file into R
- select a column and rows that fullfill a certain criterion
- generate basic graphics with annotation and added elements

Session 1: Why R, read file, select and plot data

Aim: get a general impression on how R works, without going too much into technical detail.
Afterwards, you should be able to:

- start a new script in Rstudio and execute lines of code
- read a data file into R
- select a column and rows that fullfill a certain criterion
- generate basic graphics with annotation and added elements
- open and read the documentation for R functions

Session 1: Why R, read file, select and plot data

Aim: get a general impression on how R works, without going too much into technical detail.
Afterwards, you should be able to:

- start a new script in Rstudio and execute lines of code
- read a data file into R
- select a column and rows that fullfill a certain criterion
- generate basic graphics with annotation and added elements
- open and read the documentation for R functions

-> Brief inputs followed by short exercises (for max learning)

Why R? (and not matlab, java, html, C++, fortran)

Why R? (and not matlab, java, html, C++, fortran)

R is free

Why R? (and not matlab, java, html, C++, fortran)

R is free & open source

Why R? (and not matlab, java, html, C++, fortran)

R is free & open source and has a large user community

Why R? (and not matlab, java, html, C++, fortran)

R is free & open source and has a large user community (hence many methods are implemented).

Why R? (and not matlab, java, html, C++, fortran)

R is free & open source and has a large user community (hence many methods are implemented).
It is the standard for data analysis

Why R? (and not matlab, java, html, C++, fortran)

R is free & open source and has a large user community (hence many methods are implemented).
It is the standard for data analysis and visualisation

Why R? (and not matlab, java, html, C++, fortran)

R is free & open source and has a large user community (hence many methods are implemented).
It is the standard for data analysis and visualisation in many universities and industries.

Why R? (and not matlab, java, html, C++, fortran)

R is free & open source and has a large user community (hence many methods are implemented).
It is the standard for data analysis and visualisation in many universities and industries.
R is an interpreter language

Why R? (and not matlab, java, html, C++, fortran)

R is free & open source and has a large user community (hence many methods are implemented).
It is the standard for data analysis and visualisation in many universities and industries.
R is an interpreter language $\rightarrow$ no compilation, human readable code

Why R? (and not matlab, java, html, C++, fortran)

R is free & open source and has a large user community (hence many methods are implemented).
It is the standard for data analysis and visualisation in many universities and industries.
R is an interpreter language $\rightarrow$ no compilation, human readable code
The flexibility of programming

Why R? (and not matlab, java, html, C++, fortran)

R is free & open source and has a large user community (hence many methods are implemented).
It is the standard for data analysis and visualisation in many universities and industries.
R is an interpreter language $\rightarrow$ no compilation, human readable code
The flexibility of programming enables automatization

Why R? (and not matlab, java, html, C++, fortran)

R is free & open source and has a large user community (hence many methods are implemented).
It is the standard for data analysis and visualisation in many universities and industries.
R is an interpreter language $\rightarrow$ no compilation, human readable code
The flexibility of programming enables automatization of complex statistical procedures.

Why R? (and not matlab, java, html, C++, fortran)

R is free & open source and has a large user community (hence many methods are implemented).
It is the standard for data analysis and visualisation in many universities and industries.
R is an interpreter language $\rightarrow$ no compilation, human readable code
The flexibility of programming enables automatization of complex statistical procedures.
Using R will make your work efficient,

Why R? (and not matlab, java, html, C++, fortran)

R is free & open source and has a large user community (hence many methods are implemented).
It is the standard for data analysis and visualisation in many universities and industries.
R is an interpreter language $\rightarrow$ no compilation, human readable code
The flexibility of programming enables automatization of complex statistical procedures.
Using R will make your work efficient, productive,

Why R? (and not matlab, java, html, C++, fortran)

R is free & open source and has a large user community (hence many methods are implemented).
It is the standard for data analysis and visualisation in many universities and industries.
R is an interpreter language → no compilation, human readable code
The flexibility of programming enables automatization of complex statistical procedures.
Using R will make your work efficient, productive, transparent

Why R? (and not matlab, java, html, C++, fortran)

R is free & open source and has a large user community (hence many methods are implemented).
It is the standard for data analysis and visualisation in many universities and industries.
R is an interpreter language $\rightarrow$ no compilation, human readable code
The flexibility of programming enables automatization of complex statistical procedures.
Using R will make your work efficient, productive, transparent and reproducible,

Why R? (and not matlab, java, html, C++, fortran)

R is free & open source and has a large user community (hence many methods are implemented).
It is the standard for data analysis and visualisation in many universities and industries.
R is an interpreter language → no compilation, human readable code
The flexibility of programming enables automatization of complex statistical procedures.
Using R will make your work efficient, productive, transparent and reproducible, thus giving you happiness

Why R? (and not matlab, java, html, C++, fortran)

R is free & open source and has a large user community (hence many methods are implemented).
It is the standard for data analysis and visualisation in many universities and industries.
R is an interpreter language → no compilation, human readable code
The flexibility of programming enables automatization of complex statistical procedures.
Using R will make your work efficient, productive, transparent and reproducible, thus giving you happiness and a job chance advantage.

Why R? (and not matlab, java, html, C++, fortran)

R is free & open source and has a large user community (hence many methods are implemented).
It is the standard for data analysis and visualisation in many universities and industries.
R is an interpreter language → no compilation, human readable code
The flexibility of programming enables automatization of complex statistical procedures.
Using R will make your work efficient, productive, transparent and reproducible, thus giving you happiness and a job chance advantage.

R installation instructions: bookdown.org/brry/course/install.html

Clone github repo from within Rstudio I

Install git (program for version control) bookdown.org/brry/course/git.html

Go to github.com/brry/course -> Code -> Copy URL

Clone github repo from within Rstudio II

Rstudio - File - New Project - Version Control - Git
paste the repository URL, set the subdirectory and create project.
I recommend to keep the Project directory name so local and github folder names match exactly.

From now on, get the latest version of my slides with a single click on `Pull`:

Integrated Development Environment (IDE): RStudio

Get started in R

Get started in R

Exercise 1: R is an awesome calculator

In the console, calculate 21+21 , 7*6 and $\frac{0,3}{4} * \sqrt{313600}$

If you don't know how to compute a square root in R, you can google it!

Get started in R

Exercise 1: R is an awesome calculator

In the console, calculate 21+21 , 7*6 and $\frac{0{,}3}{4} * \sqrt{313600}$

If you don't know how to compute a square root in R, you can google it!

```
21+21 ; 7*6 ; 0.3/4*sqrt(313600)
```

Get started in R

Exercise 1: R is an awesome calculator

In the console, calculate 21+21 , 7*6 and $\frac{0{,}3}{4} * \sqrt{313600}$

If you don't know how to compute a square root in R, you can google it!

```
21+21 ; 7*6 ; 0.3/4*sqrt(313600)
```

- objects: assignment $<-$ Rstudio Keyboard shortcut: `ALT` + `-`

```
nstudents <- 15
nstudents
nstudents > 12
```

Get started in R

Exercise 1: R is an awesome calculator

In the console, calculate 21+21 , 7*6 and $\frac{0{,}3}{4} * \sqrt{313600}$

If you don't know how to compute a square root in R, you can google it!

```
21+21 ; 7*6 ; 0.3/4*sqrt(313600)
```

- objects: assignment $<-$ Rstudio Keyboard shortcut: `ALT` + `-`

```
nstudents <- 15
nstudents
nstudents > 12
```

- What's a good object name?

Get started in R

Exercise 1: R is an awesome calculator

In the console, calculate 21+21 , 7*6 and $\frac{0{,}3}{4} * \sqrt{313600}$

If you don't know how to compute a square root in R, you can google it!

```
21+21 ; 7*6 ; 0.3/4*sqrt(313600)
```

- objects: assignment $<-$ Rstudio Keyboard shortcut: `ALT` + `-`

```
nstudents <- 15
nstudents
nstudents > 12
```

- What's a good object name? $\rightarrow$ short, but explanatory,

Get started in R

Exercise 1: R is an awesome calculator

In the console, calculate 21+21 , 7*6 and $\frac{0{,}3}{4} * \sqrt{313600}$

If you don't know how to compute a square root in R, you can google it!

```
21+21 ; 7*6 ; 0.3/4*sqrt(313600)
```

- objects: assignment $<-$ Rstudio Keyboard shortcut: `ALT` + `-`

```
nstudents <- 15
nstudents
nstudents > 12
```

- What's a good object name? $\rightarrow$ short, but explanatory, lowerCamelStandard_or_underscore are good naming conventions

Get started in R

Exercise 1: R is an awesome calculator

In the console, calculate 21+21 , 7*6 and $\frac{0{,}3}{4} * \sqrt{313600}$

If you don't know how to compute a square root in R, you can google it!

```
21+21 ; 7*6 ; 0.3/4*sqrt(313600)
```

- objects: assignment $<-$ Rstudio Keyboard shortcut: `ALT` + `-`

```
nstudents <- 15
nstudents
nstudents > 12
```

- What's a good object name? → short, but explanatory, lowerCamelStandard_or_underscore are good naming conventions
- comments: `# everything after a hashtag is not executed.`

Get started in R

Exercise 1: R is an awesome calculator

In the console, calculate 21+21 , 7*6 and $\frac{0{,}3}{4} * \sqrt{313600}$

If you don't know how to compute a square root in R, you can google it!

```
21+21 ; 7*6 ; 0.3/4*sqrt(313600)
```

- objects: assignment $<-$ Rstudio Keyboard shortcut: `ALT` + `-`

```
nstudents <- 15
nstudents
nstudents > 12
```

- What's a good object name? → short, but explanatory, lowerCamelStandard_or_underscore are good naming conventions
- comments: `# everything after a hashtag is not executed.`
- Rstudio scripts: Send line/selection to R console: `CTRL` + `ENTER`

Objects: vectors

- To create a vector with values, use the syntax
  `values <- c(3, -11, 13, 5.94) # concatenate, combine`

Objects: vectors

- To create a vector with values, use the syntax
  `values <- c(3, -11, 13, 5.94) # concatenate, combine`
- To obtain a subset (=indexing), use square brackets: `values[2]`

Objects: vectors

- To create a vector with values, use the syntax
  `values <- c(3, -11, 13, 5.94) # concatenate, combine`
- To obtain a subset (=indexing), use square brackets: `values[2]`

Exercise 2: Vector indexing

1. Create a vector with body sizes of people around you. You can also use the values 1.75, 1.76, 1.83, 1.84, 1.77, 1.76, 1.77, 1.66, 1.86, 1.76. Assign it to an object with a useful name (`YourObject` is not one!).
2. What does `3:6` create? What does `YourObject[3:6]` do?
3. What does `YourObject[-4]` do?
4. BONUS (for fast people): Analyze the descriptive statistics: `mean(YourObject), median, min, max, range, quantile`
5. BONUS 2: Help your peers - there's no reason to be bored ;-)

Vector task solutions

```
size <- c(1.75, 1.76, 1.83, 1.84, 1.77, 1.76,
          1.77, 1.66, 1.86, 1.76)
3:6 # A vector with consecutive integers
size[3:6] # Select the corresponding elements of a vector
size[-4] # Select all but the fourth value
```

```
c(mean(size), median(size), min(size), max(size)); range(size); quantile(size)
```

```
## [1] 1.776 1.765 1.660 1.860
## [1] 1.66 1.86
##    0%   25%   50%   75%  100%
## 1.660 1.760 1.765 1.815 1.860
```

`mean` and `median` are similar, hinting at a symmetric data distribution
`range` returns a composite of `min`imum and `max`imum
`quantile` returns quantiles (now that is surprising...)

Outline

R course info
1. One-Session-Intro
   1. Intro, objects, vectors
   2. Files, data.frames
   3. Plotting
   4. Packages, regression
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Reading files

Reading files

- Read files into R with the command `read.table`

Reading files

- Read files into R with the command `read.table`
- `?read.table`, or press F1 to read the documentation

Reading files

- Read files into R with the command `read.table`
- `?read.table`, or press F1 to read the documentation

In case you did not clone the course repo:

- Download treesize.txt *(rightclick **Raw**, save as)*
- Tell R (in the script) where to look for files with:
  `setwd("C:/path/to/input") # use forwardslashes`

Reading files

- Read files into R with the command `read.table`
- `?read.table`, or press F1 to read the documentation

In case you did not clone the course repo:

- Download treesize.txt *(rightclick **Raw**, save as)*
- Tell R (in the script) where to look for files with:
  `setwd("C:/path/to/input") # use forwardslashes`

Exercise 3: Reading files

Reading files

- Read files into R with the command `read.table`
- `?read.table`, or press F1 to read the documentation

In case you did not clone the course repo:

- Download treesize.txt *(rightclick **Raw**, save as)*
- Tell R (in the script) where to look for files with:
  `setwd("C:/path/to/input") # use forwardslashes`

Exercise 3: Reading files

- Read the file into R with the command `read.table`, assigning it to an object with a good name.

Reading files

- Read files into R with the command `read.table`
- `?read.table`, or press F1 to read the documentation

In case you did not clone the course repo:

- Download treesize.txt *(rightclick **Raw**, save as)*
- Tell R (in the script) where to look for files with:
  `setwd("C:/path/to/input") # use forwardslashes`

Exercise 3: Reading files

- Read the file into R with the command `read.table`, assigning it to an object with a good name.
- If R tells you "no such file" exists, check the output of `dir()`.

Reading files

- Read files into R with the command `read.table`
- `?read.table`, or press F1 to read the documentation

In case you did not clone the course repo:

- Download treesize.txt *(rightclick **Raw**, save as)*
- Tell R (in the script) where to look for files with:
  `setwd("C:/path/to/input") # use forwardslashes`

Exercise 3: Reading files

- Read the file into R with the command `read.table`, assigning it to an object with a good name.
- If R tells you "no such file" exists, check the output of `dir()`.
- Check the object with `head(YourObject)`.

Reading files

- Read files into R with the command `read.table`
- `?read.table`, or press F1 to read the documentation

In case you did not clone the course repo:

- Download treesize.txt *(rightclick **Raw**, save as)*
- Tell R (in the script) where to look for files with:
  `setwd("C:/path/to/input") # use forwardslashes`

Exercise 3: Reading files

- Read the file into R with the command `read.table`, assigning it to an object with a good name.
- If R tells you "no such file" exists, check the output of `dir()`.
- Check the object with `head(YourObject)`.
- `str(YourObject)` must yield the column data types: `num, num, factor`.

Reading files

- Read files into R with the command `read.table`
- `?read.table`, or press F1 to read the documentation

In case you did not clone the course repo:

- Download treesize.txt *(rightclick **Raw**, save as)*
- Tell R (in the script) where to look for files with:
  `setwd("C:/path/to/input") # use forwardslashes`

Exercise 3: Reading files

- Read the file into R with the command `read.table`, assigning it to an object with a good name.
- If R tells you "no such file" exists, check the output of `dir()`.
- Check the object with `head(YourObject)`.
- `str(YourObject)` must yield the column data types: `num, num, factor`.
- You need to set the argument `header`.

Solution to exercise 3: Reading files

```
treesize <- read.table(file="treesize.txt", header=TRUE)
str(treesize)
```

Objects: data.frames

- For tables with different data types (numbers, characters, categories, integers), R has the object type data.frame:
  `data.frame(count=c(2,6,5), type=c("a","k","k"))`

Objects: data.frames

- For tables with different data types (numbers, characters, categories, integers), R has the object type data.frame:
  `data.frame(count=c(2,6,5), type=c("a","k","k"))`
- `read.table` also returns a data.frame

Objects: data.frames

- For tables with different data types (numbers, characters, categories, integers), R has the object type data.frame:
  `data.frame(count=c(2,6,5), type=c("a","k","k"))`
- `read.table` also returns a data.frame
- If we have the object `DF`, we can subset with `DF[rows,columns]`

Objects: data.frames

- For tables with different data types (numbers, characters, categories, integers), R has the object type data.frame:
  `data.frame(count=c(2,6,5), type=c("a","k","k"))`
- `read.table` also returns a data.frame
- If we have the object `DF`, we can subset with `DF[rows,columns]`
- `DF[1,2:4];`

Objects: data.frames

- For tables with different data types (numbers, characters, categories, integers), R has the object type data.frame:
  `data.frame(count=c(2,6,5), type=c("a","k","k"))`
- `read.table` also returns a data.frame
- If we have the object `DF`, we can subset with `DF[rows,columns]`
- `DF[1,2:4]; DF[2, ];`

Objects: data.frames

- For tables with different data types (numbers, characters, categories, integers), R has the object type data.frame:
  `data.frame(count=c(2,6,5), type=c("a","k","k"))`
- `read.table` also returns a data.frame
- If we have the object `DF`, we can subset with `DF[rows,columns]`
- `DF[1,2:4]; DF[2, ];  DF[ ,"name"];`

Objects: data.frames

- For tables with different data types (numbers, characters, categories, integers), R has the object type data.frame:
  `data.frame(count=c(2,6,5), type=c("a","k","k"))`
- `read.table` also returns a data.frame
- If we have the object `DF`, we can subset with `DF[rows,columns]`
- `DF[1,2:4]; DF[2, ];  DF[ ,"name"]; DF$name`

Objects: data.frames

- For tables with different data types (numbers, characters, categories, integers), R has the object type data.frame:
  `data.frame(count=c(2,6,5), type=c("a","k","k"))`
- `read.table` also returns a data.frame
- If we have the object `DF`, we can subset with `DF[rows,columns]`
- `DF[1,2:4]; DF[2, ];  DF[ ,"name"]; DF$name`
- Logical values: `vect[c(TRUE,TRUE,FALSE,FALSE,TRUE,FALSE)]`

Objects: data.frames

- For tables with different data types (numbers, characters, categories, integers), R has the object type data.frame:
  `data.frame(count=c(2,6,5), type=c("a","k","k"))`
- `read.table` also returns a data.frame
- If we have the object `DF`, we can subset with `DF[rows,columns]`
- `DF[1,2:4]; DF[2, ];  DF[ ,"name"]; DF$name`
- Logical values: `vect[c(TRUE,TRUE,FALSE,FALSE,TRUE,FALSE)]`

Exercise 4: Data.frame indexing

From the dataset `treesize` from the previous exercise, obtain:

1. The first 5 values in column 2
2. The maximum "Height" (the maximum of the values in that column)
3. For each entry: is the measurement equal to (==) A?
4. BONUS 1: The height entries for trees older than 23.5 years
5. BONUS 2: All rows, excluding rows 3,7,8,9,...,90

Solution to exercise 4: subsetting data.frames I

```
treesize[1:5, 2]

## [1] 32.7 26.0 27.9 35.4 26.4

max(treesize$height)  ;   max(treesize[ ,"age"])

## [1] 40.1
## [1] 24.8

treesize$measurement=="A"

##   [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
##  [13]  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE  TRUE
##  [25]  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE
##  [37] FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE
##  [49]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE
##  [61]  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE
##  [73] FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE
##  [85] FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE
##  [97]  TRUE FALSE  TRUE  TRUE
```

Solution to exercise 4: subsetting data.frames II

```
treesize[ treesize$age >= 23.5 , "height"]

##  [1] 32.7 38.4 40.1 34.0 32.3 32.6 34.4 23.2 34.6 35.9
```

```
treesize[ -c(3,7:90) , ]

##     age height measurement
## 1  24.4   32.7           A
## 2  13.6   26.0           A
## 4  21.2   35.4           A
## 5  19.8   26.4           B
## 6   7.7    7.2           B
## 91 23.1   39.0           A
## 92 24.6   34.6           A
## 93  7.7    9.2           B
## 94 10.1   10.3           A
## 95 23.8   35.9           A
## 96  7.7    7.4           B
## 97 17.8   31.6           A
## ...
```

Outline

R course info
1. One-Session-Intro
   1. Intro, objects, vectors
   2. Files, data.frames
   3. Plotting
   4. Packages, regression
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Plotting I

General code for scatterplots: `plot(x, y, ...)`

Plotting I

General code for scatterplots: `plot(x, y, ...)`

```
plot(x=Indometh$time, y=Indometh$conc,
     col="orange", pch=16, main="Awesome Graph!")
```

Plotting I

General code for scatterplots: `plot(x, y, ...)`

```
points(4, 1.5, pch=22, bg="yellow", cex=4, col="red")
# PointCHaracter, BackGround, Character EXpansion
```

Plotting I

General code for scatterplots: `plot(x, y, ...)`

```
lines(x=c(2,5,6,7), y=c(1,2.3,-3,1),
      col="blue", type="b", lwd=5)
```

Plotting II: Our treesize dataset

```
treesize <- read.table(file="data/treesize.txt", header=TRUE)
```

Plotting II: Our treesize dataset

```
treesize <- read.table(file="data/treesize.txt", header=TRUE)
```

```
plot(x=xvalues, y=y_values, xlab="nice axis label",
     main="graph title", las=1)
```

Plotting II: Our treesize dataset

```
treesize <- read.table(file="data/treesize.txt", header=TRUE)
```

```
plot(x=xvalues, y=y_values, xlab="nice axis label",
     main="graph title", las=1)
```

Exercise 5: Scatterplots, sample distribution graphs

1. Plot tree height over age.
2. Add labels to the plot.
3. Change the point character (`pch`) and color (`col`).
4. BONUS 1: Use a catogory vector for colors, e.g. as.factor(treesize$measurement)
5. BONUS 2: Compare the histogram (`hist`) of the heights with the `boxplot` and `quantile(x, probs=c(0.1, 0.8))`.

Solution for exercise 5: Scatterplots

```
treesize <- read.table(file="data/treesize.txt", header=TRUE)

plot(treesize$age, treesize$height)
plot(treesize$age, treesize$height, las=1, ylab="Tree height [m]",
     xlab="Tree age [years]", col=as.factor(treesize$measurement),
     main="Older trees are larger", pch=3)
quantile(treesize$height, probs=c(0.1, 0.8))

##   10%   80%
##  8.93 32.36
```

Solution for exercise 5: Scatterplot

Solution for exercise 5: Histogram

```
hist(treesize$height, col="purple", breaks=20, las=1)
```

Solution for exercise 5: Boxplot

```
boxplot(treesize$height, col="cyan", horizontal=TRUE,
        notch=TRUE)
```

commonly needed plot arguments

```
plot(x, y, # point coordinates
col="lightblue", # point color
pch=0, # point character (symbol)
xlab="My label  [km]", ylab="", # axis labels
main="Graph title", # title
cex=1.8, # character expansion (symbol size)
type="l", # draw lines instead of points
lwd=3, # line width (thickness of lines)
las=1, # label axis style (axis numbers upright)
xaxt="n" # axis type (none to suppress axis)
)
```

Outline

R course info
1. One-Session-Intro
   1. Intro, objects, vectors
   2. Files, data.frames
   3. Plotting
   4. Packages, regression
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

R Packages

- Many people write code for specific tasks and publish it on CRAN, the Comprehensive R Archive Network

R Packages

- Many people write code for specific tasks and publish it on CRAN, the Comprehensive R Archive Network
- Packages for a range of topics: cran.r-project.org/web/views

R Packages

- Many people write code for specific tasks and publish it on CRAN, the Comprehensive R Archive Network
- Packages for a range of topics: cran.r-project.org/web/views
- All >16'000 available packages: cran.r-project.org/web/packages

R Packages

- Many people write code for specific tasks and publish it on CRAN, the Comprehensive R Archive Network
- Packages for a range of topics: cran.r-project.org/web/views
- All >16'000 available packages: cran.r-project.org/web/packages
- `install.packages("ggplot2")` to download and install.
  (only needs to be executed once, works on user level, no admin rights required)

  You can do this in Rstudio

R Packages

- Many people write code for specific tasks and publish it on CRAN, the Comprehensive R Archive Network
- Packages for a range of topics: cran.r-project.org/web/views
- All >16'000 available packages: cran.r-project.org/web/packages
- `install.packages("ggplot2")` to download and install.
  (only needs to be executed once, works on user level, no admin rights required)

  You can do this in Rstudio
- `library("ggplot2")` to load it

  (needed in every new R session) Put this in the script for reproducibility

R Packages

- Many people write code for specific tasks and publish it on CRAN, the Comprehensive R Archive Network
- Packages for a range of topics: cran.r-project.org/web/views
- All >16'000 available packages: cran.r-project.org/web/packages
- `install.packages("ggplot2")` to download and install.
  (only needs to be executed once, works on user level, no admin rights required)

  You can do this in Rstudio
- `library("ggplot2")` to load it

  (needed in every new R session) Put this in the script for reproducibility
- Better to use the `package::function` syntax

R Packages

- Many people write code for specific tasks and publish it on CRAN, the Comprehensive R Archive Network
- Packages for a range of topics: cran.r-project.org/web/views
- All >16'000 available packages: cran.r-project.org/web/packages
- `install.packages("ggplot2")` to download and install.
  (only needs to be executed once, works on user level, no admin rights required)

  You can do this in Rstudio
- `library("ggplot2")` to load it

  (needed in every new R session) Put this in the script for reproducibility
- Better to use the `package::function` syntax
- Regularly run `update.packages()` or use the Rstudio button

R Packages

- Many people write code for specific tasks and publish it on CRAN, the Comprehensive R Archive Network
- Packages for a range of topics: cran.r-project.org/web/views
- All >16'000 available packages: cran.r-project.org/web/packages
- `install.packages("ggplot2")` to download and install.
  (only needs to be executed once, works on user level, no admin rights required)
  You can do this in Rstudio
- `library("ggplot2")` to load it
  (needed in every new R session) Put this in the script for reproducibility
- Better to use the `package::function` syntax
- Regularly run `update.packages()` or use the Rstudio button
- Rarely needed: `remove.packages("packagename")`

Packages and linear regression

Exercise 6: Packages, linear regression

1. Install and load the package `berryFunctions`
2. Pass the treesize.txt *(rightclick **Raw**, save as)* data to `linReg`. Don't use (x,y) coordinates, but a (y~x) formula, read as "y dependent on x". Use the `data` argument as described in the "Examples" section of the `linReg` manual. Describe the resulting graph (height vs age).
3. Feed the data into `lm`, assign the output to an object (useful name!).
4. Briefly explain the `summary` of the linear model.
5. BONUS1: Look into the source code of `linReg`. You can print the function by calling it without brackets. You can also open it on github with `berryFunctions::funSource`.
6. BONUS2: What is the backbone for the calculation in `linReg`?
7. BONUS3: install rskey, set a keyboard shortcut for `funSource` and use it to get the source code of some function.

Solution for exercise 6: Linear regression

```
library("berryFunctions")
linReg(height~age, data=treesize)
linReg
funSource(linReg) # berryFunctions/R/linReg.R  -  working horse: lm
linear_model <- lm(height~age, data=treesize)
summary(linear_model)
```

blog.yhat.com/posts/r-lm-summary.html

stats.stackexchange.com/questions/5135/interpretation-of-rs-lm-output

Session 1: Recap (They tried to make me go to recap and I said, 'Yes, yes, yes.')

Session 1: Recap (They tried to make me go to recap and I said, 'Yes, yes, yes.')

- R is awesome. You're here, thus you will be R-some, too

Session 1: Recap (They tried to make me go to recap and I said, 'Yes, yes, yes.')

- R is awesome. You're here, thus you will be R-some, too
- Console for code execution (R), ready with $>$, waiting with $+$

Session 1: Recap (They tried to make me go to recap and I said, 'Yes, yes, yes.')

- R is awesome. You're here, thus you will be R-some, too
- Console for code execution (R), ready with $>$, waiting with $+$
- Script for code writing (you) + exchange

Session 1: Recap (They tried to make me go to recap and I said, 'Yes, yes, yes.')

- R is awesome. You're here, thus you will be R-some, too
- Console for code execution (R), ready with `>`, waiting with `+`
- Script for code writing (you) + exchange
- Objects are created with the assignment operator `<-`

Session 1: Recap (They tried to make me go to recap and I said, 'Yes, yes, yes.')

- R is awesome. You're here, thus you will be R-some, too
- Console for code execution (R), ready with `>`, waiting with `+`
- Script for code writing (you) + exchange
- Objects are created with the assignment operator `<-`
- Get help with `help(median)`, `?matrix`, or pressing F1

Session 1: Recap (They tried to make me go to recap and I said, 'Yes, yes, yes.')

- R is awesome. You're here, thus you will be R-some, too
- Console for code execution (R), ready with `>`, waiting with `+`
- Script for code writing (you) + exchange
- Objects are created with the assignment operator `<-`
- Get help with `help(median)`, `?matrix`, or pressing F1
- Create vectors with `c(42,3,-4)`, `1:n`

Session 1: Recap (They tried to make me go to recap and I said, 'Yes, yes, yes.')

- R is awesome. You're here, thus you will be R-some, too
- Console for code execution (R), ready with `>`, waiting with `+`
- Script for code writing (you) + exchange
- Objects are created with the assignment operator `<-`
- Get help with `help(median)`, `?matrix`, or pressing F1
- Create vectors with `c(42,3,-4)`, `1:n`
- `data.frame`s hold tables with one data type per column

Session 1: Recap (They tried to make me go to recap and I said, 'Yes, yes, yes.')

- R is awesome. You're here, thus you will be R-some, too
- Console for code execution (R), ready with `>`, waiting with `+`
- Script for code writing (you) + exchange
- Objects are created with the assignment operator `<-`
- Get help with `help(median)`, `?matrix`, or pressing F1
- Create vectors with `c(42,3,-4)`, `1:n`
- `data.frame`s hold tables with one data type per column
- `read.table`, with `header=TRUE` to read files into R

Session 1: Recap (They tried to make me go to recap and I said, 'Yes, yes, yes.')

- R is awesome. You're here, thus you will be R-some, too
- Console for code execution (R), ready with `>`, waiting with `+`
- Script for code writing (you) + exchange
- Objects are created with the assignment operator `<-`
- Get help with `help(median)`, `?matrix`, or pressing F1
- Create vectors with `c(42,3,-4)`, `1:n`
- `data.frame`s hold tables with one data type per column
- `read.table`, with `header=TRUE` to read files into R
- Index objects with `values[8:5]`, `tab[4,2:6]`

Session 1: Recap (They tried to make me go to recap and I said, 'Yes, yes, yes.')

- R is awesome. You're here, thus you will be R-some, too
- Console for code execution (R), ready with `>`, waiting with `+`
- Script for code writing (you) + exchange
- Objects are created with the assignment operator `<-`
- Get help with `help(median)`, `?matrix`, or pressing F1
- Create vectors with `c(42,3,-4)`, `1:n`
- `data.frame`s hold tables with one data type per column
- `read.table`, with `header=TRUE` to read files into R
- Index objects with `values[8:5]`, `tab[4,2:6]`
- df columns with `tab[ ,3]`, `tab[ ,"precip"]`, `tab$wind`

Session 1: Recap (They tried to make me go to recap and I said, 'Yes, yes, yes.')

- R is awesome. You're here, thus you will be R-some, too
- Console for code execution (R), ready with `>`, waiting with `+`
- Script for code writing (you) + exchange
- Objects are created with the assignment operator `<-`
- Get help with `help(median)`, `?matrix`, or pressing F1
- Create vectors with `c(42,3,-4)`, `1:n`
- `data.frame`s hold tables with one data type per column
- `read.table`, with `header=TRUE` to read files into R
- Index objects with `values[8:5]`, `tab[4,2:6]`
- df columns with `tab[ ,3]`, `tab[ ,"precip"]`, `tab$wind`
- `hist(vals)`; `plot(x,y, las=1, pch=16, type="b")`

Session 1: Recap (They tried to make me go to recap and I said, 'Yes, yes, yes.')

- R is awesome. You're here, thus you will be R-some, too
- Console for code execution (R), ready with `>`, waiting with `+`
- Script for code writing (you) + exchange
- Objects are created with the assignment operator `<-`
- Get help with `help(median)`, `?matrix`, or pressing F1
- Create vectors with `c(42,3,-4)`, `1:n`
- `data.frame`s hold tables with one data type per column
- `read.table`, with `header=TRUE` to read files into R
- Index objects with `values[8:5]`, `tab[4,2:6]`
- df columns with `tab[ ,3]`, `tab[ ,"precip"]`, `tab$wind`
- `hist(vals)`; `plot(x,y, las=1, pch=16, type="b")`

Session 1: Recap (They tried to make me go to recap and I said, 'Yes, yes, yes.')

- R is awesome. You're here, thus you will be R-some, too
- Console for code execution (R), ready with `>`, waiting with `+`
- Script for code writing (you) + exchange
- Objects are created with the assignment operator `<-`
- Get help with `help(median)`, `?matrix`, or pressing F1
- Create vectors with `c(42,3,-4)`, `1:n`
- `data.frame`s hold tables with one data type per column
- `read.table`, with `header=TRUE` to read files into R
- Index objects with `values[8:5]`, `tab[4,2:6]`
- df columns with `tab[ ,3]`, `tab[ ,"precip"]`, `tab$wind`
- `hist(vals)`; `plot(x,y, las=1, pch=16, type="b")`

Homework: finish all covered exercises, work on exercise 7 (Reading files, subsetting, comparing)

Time to put all of this into practice

Exercise 7: Reading files, subsetting, comparing

1. Start a new session of R. Read in the tree size data.
2. Check that everything is read correctly with `str`.
3. `which` rows of the data.frame have measurement equal to (`==`) B?
4. Plot the linear regression for measurement method B, then add the points and regression line for A in a different color and shape.
5. BONUS 1: use `boxplot` with a formula (`height~measurement`) for an automatic boxplot comparison with median difference notches.
6. BONUS 2: Visually compare the effect of Girth and Height on Volume in the dataset `trees`. Here's one idea (of many possible): Plot with two panels below each other (`par(mfrow=c(2,1))`), with the linear regression plots (`berryFunctions::linReg`) for each variable. Make it look nice with the graphical `par`ameters `mar` and `mgp`, as well as by specifying some of the linReg arguments.

Solution for exercise 7: Reading files, subsetting

```r
setwd("D:/my/path")
treesize <- read.table(file="data/treesize.txt", header=TRUE)
str(treesize)
```

```r
which(treesize$measurement=="B")

##  [1]  5  6 12 16 18 19 20 22 26 28 29 30 31 34 35
## [16] 36 37 39 41 42 46 47 48 50 55 56 59 60 63 64
## [31] 66 68 70 71 72 73 74 75 76 78 80 81 85 86 87
## [46] 88 90 93 96 98

treeA <- treesize[treesize$measurement=="A", ]
treeB <- treesize[treesize$measurement=="B", ]
library("berryFunctions")
linReg(height~age, data=treeA, pos1="topleft")
linReg(height~age, data=treeB, add=T, colline="blue",
       pos1="bottomright", inset=0.05)
points(height~age, data=treeB, pch=3, col="blue")
```

Solution for exercise 7: Plotting, linear Regression

Solution for exercise 7.BONUS 1

```
boxplot(height~measurement, data=treesize,
        col=c("blue","orange"), notch=TRUE)
```

Solution for exercise 7.BONUS 2

```
par(mfrow=c(2,1), mar=c(3,3,1.5,0.5), mgp=c(1.9,0.7,0), las=1)
linReg(Volume~Height, data=trees, col="blue", pos1="topleft")
linReg(Volume~Girth, data=trees, col="forestgreen", pos1="topleft")
```

Outline

R course info
1. One-Session-Intro
2. Objects
   1. Rstudio + getting help
   2. R syntax
   3. Assignments
   4. Vectors, indexing
   5. Data.frames
   6. Matrices, apply function
   7. Lists, lapply/sapply functions
   8. Classes and methods, data types and object types
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

RStudio overview

RStudio configuration

Recommended settings for reproducible research under
Rstudio - Tools - Global Options - General
OFF: Restore .Rdata into workspace at startup
Save workspace to .RData on exit: **NEVER**

RStudio configuration

Recommended settings for reproducible research under
Rstudio - Tools - Global Options - General
OFF: Restore .Rdata into workspace at startup
Save workspace to .RData on exit: **NEVER**

Instead use `save(object, file="object.Rdata")` *after long computations. You can load them later with* `load("object.Rdata")`.

RStudio configuration

Recommended settings for reproducible research under
Rstudio - Tools - Global Options - General
OFF: Restore .Rdata into workspace at startup
Save workspace to .RData on exit: **NEVER**

Instead use `save(object, file="object.Rdata")` *after long computations. You can load them later with* `load("object.Rdata")`.

More at bookdown.org/brry/course/settings.html

RStudio configuration

Recommended settings for reproducible research under
Rstudio - Tools - Global Options - General
OFF: Restore .Rdata into workspace at startup
Save workspace to .RData on exit: **NEVER**

Instead use `save(object, file="object.Rdata")` *after long computations. You can load them later with* `load("object.Rdata")`.

More at bookdown.org/brry/course/settings.html

keyboard shortcuts (ALT + SHIFT + K)

RStudio configuration

Recommended settings for reproducible research under
Rstudio - Tools - Global Options - General
OFF: Restore .Rdata into workspace at startup
Save workspace to .RData on exit: **NEVER**

Instead use `save(object, file="object.Rdata")` *after long computations. You can load them later with* `load("object.Rdata")`.

More at bookdown.org/brry/course/settings.html

keyboard shortcuts (`ALT` + `SHIFT` + `K`)

1. `CTRL` + `ENTER` (in source script) send line/selection to console

RStudio configuration

Recommended settings for reproducible research under
Rstudio - Tools - Global Options - General
OFF: Restore .Rdata into workspace at startup
Save workspace to .RData on exit: **NEVER**

Instead use `save(object, file="object.Rdata")` *after long computations. You can load them later with* `load("object.Rdata")`.

More at bookdown.org/brry/course/settings.html

keyboard shortcuts (`ALT` + `SHIFT` + `K`)

1. `CTRL` + `ENTER` (in source script) send line/selection to console
2. `CTRL` + `UP` (in console) command history

RStudio configuration

Recommended settings for reproducible research under
Rstudio - Tools - Global Options - General
OFF: Restore .Rdata into workspace at startup
Save workspace to .RData on exit: **NEVER**

Instead use `save(object, file="object.Rdata")` *after long computations. You can load them later with* `load("object.Rdata")`.

More at bookdown.org/brry/course/settings.html

keyboard shortcuts (`ALT` + `SHIFT` + `K`)

1. `CTRL` + `ENTER` (in source script) send line/selection to console
2. `CTRL` + `UP` (in console) command history
3. `CTRL` + `SHIFT` + `N` new R script

RStudio configuration

Recommended settings for reproducible research under
Rstudio - Tools - Global Options - General
OFF: Restore .Rdata into workspace at startup
Save workspace to .RData on exit: **NEVER**

Instead use `save(object, file="object.Rdata")` *after long computations. You can load them later with* `load("object.Rdata")`.

More at bookdown.org/brry/course/settings.html

keyboard shortcuts (`ALT` + `SHIFT` + `K`)

1. `CTRL` + `ENTER` (in source script) send line/selection to console
2. `CTRL` + `UP` (in console) command history
3. `CTRL` + `SHIFT` + `N` new R script
4. `CTRL` + `O` open file

RStudio configuration

Recommended settings for reproducible research under
Rstudio - Tools - Global Options - General
OFF: Restore .Rdata into workspace at startup
Save workspace to .RData on exit: **NEVER**

Instead use `save(object, file="object.Rdata")` *after long computations. You can load them later with* `load("object.Rdata")`.

More at bookdown.org/brry/course/settings.html

keyboard shortcuts (`ALT` + `SHIFT` + `K`)

1. `CTRL` + `ENTER` (in source script) send line/selection to console
2. `CTRL` + `UP` (in console) command history
3. `CTRL` + `SHIFT` + `N` new R script
4. `CTRL` + `O` open file
5. `CTRL` + `S` save script

RStudio configuration

Recommended settings for reproducible research under
Rstudio - Tools - Global Options - General
OFF: Restore .Rdata into workspace at startup
Save workspace to .RData on exit: **NEVER**

Instead use `save(object, file="object.Rdata")` *after long computations. You can load them later with* `load("object.Rdata")`.

More at bookdown.org/brry/course/settings.html

keyboard shortcuts (`ALT` + `SHIFT` + `K`)

1. `CTRL` + `ENTER` (in source script) send line/selection to console
2. `CTRL` + `UP` (in console) command history
3. `CTRL` + `SHIFT` + `N` new R script
4. `CTRL` + `O` open file
5. `CTRL` + `S` save script
6. `CTRL` + `TAB` next Tab

RStudio configuration

Recommended settings for reproducible research under
Rstudio - Tools - Global Options - General
OFF: Restore .Rdata into workspace at startup
Save workspace to .RData on exit: **NEVER**

Instead use `save(object, file="object.Rdata")` *after long computations. You can load them later with* `load("object.Rdata")`.

More at bookdown.org/brry/course/settings.html

keyboard shortcuts (`ALT` + `SHIFT` + `K`)

1. `CTRL` + `ENTER` (in source script) send line/selection to console
2. `CTRL` + `UP` (in console) command history
3. `CTRL` + `SHIFT` + `N` new R script
4. `CTRL` + `O` open file
5. `CTRL` + `S` save script
6. `CTRL` + `TAB` next Tab
7. `CTRL` + `SHIFT` + `TAB` previous Tab

Rstudio Tips and Tricks

1. RStudio IDE Easy Tricks You Might've Missed and rviews.rstudio.com/categories/tips-and-tricks

Rstudio Tips and Tricks

1. RStudio IDE Easy Tricks You Might've Missed and rviews.rstudio.com/categories/tips-and-tricks
2. Work with Rstudio projects!

Rstudio Tips and Tricks

1. RStudio IDE Easy Tricks You Might've Missed and rviews.rstudio.com/categories/tips-and-tricks
2. Work with Rstudio projects!
3. `ALT` + mouse for multiline cursor. *or*: `CTRL` + `ALT` + `UP` / `DOWN`

Rstudio Tips and Tricks

1. RStudio IDE Easy Tricks You Might've Missed and rviews.rstudio.com/categories/tips-and-tricks
2. Work with Rstudio projects!
3. `ALT` + mouse for multiline cursor. *or*: `CTRL` + `ALT` + `UP` / `DOWN`
4. `CTRL` + `1` cursor to panel (1:source, 2:console, 3:help, 4:history, 5:files, 6:plot...)

Rstudio Tips and Tricks

1. RStudio IDE Easy Tricks You Might've Missed and rviews.rstudio.com/categories/tips-and-tricks
2. Work with Rstudio projects!
3. `ALT` + mouse for multiline cursor. *or*: `CTRL` + `ALT` + `UP` / `DOWN`
4. `CTRL` + `1` cursor to panel (1:source, 2:console, 3:help, 4:history, 5:files, 6:plot...)
5. `CTRL` + `SHIFT` + `1` / `2` / `3` / `...` panel full view

Rstudio Tips and Tricks

1. RStudio IDE Easy Tricks You Might've Missed and rviews.rstudio.com/categories/tips-and-tricks
2. Work with Rstudio projects!
3. `ALT` + mouse for multiline cursor. *or*: `CTRL` + `ALT` + `UP` / `DOWN`
4. `CTRL` + `1` cursor to panel (1:source, 2:console, 3:help, 4:history, 5:files, 6:plot...)
5. `CTRL` + `SHIFT` + `1` / `2` / `3` / `...` panel full view
6. `ALT` + `UP` / `DOWN` move line of code

Rstudio Tips and Tricks

1. RStudio IDE Easy Tricks You Might've Missed and rviews.rstudio.com/categories/tips-and-tricks
2. Work with Rstudio projects!
3. `ALT` + mouse for multiline cursor. *or*: `CTRL` + `ALT` + `UP` / `DOWN`
4. `CTRL` + `1` cursor to panel (1:source, 2:console, 3:help, 4:history, 5:files, 6:plot...)
5. `CTRL` + `SHIFT` + `1` / `2` / `3` / `...` panel full view
6. `ALT` + `UP` / `DOWN` move line of code
7. `CTRL` + `SHIFT` + `P` rerun previous code region

Rstudio Tips and Tricks

1. RStudio IDE Easy Tricks You Might've Missed and rviews.rstudio.com/categories/tips-and-tricks
2. Work with Rstudio projects!
3. `ALT` + mouse for multiline cursor. *or*: `CTRL` + `ALT` + `UP` / `DOWN`
4. `CTRL` + `1` cursor to panel (1:source, 2:console, 3:help, 4:history, 5:files, 6:plot...)
5. `CTRL` + `SHIFT` + `1` / `2` / `3` / `...` panel full view
6. `ALT` + `UP` / `DOWN` move line of code
7. `CTRL` + `SHIFT` + `P` rerun previous code region
8. `CTRL` + `SHIFT` + `S` / `ENTER` source document

Rstudio Tips and Tricks

1. RStudio IDE Easy Tricks You Might've Missed and rviews.rstudio.com/categories/tips-and-tricks
2. Work with Rstudio projects!
3. `ALT` + mouse for multiline cursor. *or*: `CTRL` + `ALT` + `UP` / `DOWN`
4. `CTRL` + `1` cursor to panel (1:source, 2:console, 3:help, 4:history, 5:files, 6:plot...)
5. `CTRL` + `SHIFT` + `1` / `2` / `3` / `...` panel full view
6. `ALT` + `UP` / `DOWN` move line of code
7. `CTRL` + `SHIFT` + `P` rerun previous code region
8. `CTRL` + `SHIFT` + `S` / `ENTER` source document
9. `CTRL` + `SHIFT` + `C` (un)comment lines

Rstudio Tips and Tricks

1. RStudio IDE Easy Tricks You Might've Missed and rviews.rstudio.com/categories/tips-and-tricks
2. Work with Rstudio projects!
3. `ALT` + mouse for multiline cursor. *or*: `CTRL` + `ALT` + `UP` / `DOWN`
4. `CTRL` + `1` cursor to panel (1:source, 2:console, 3:help, 4:history, 5:files, 6:plot...)
5. `CTRL` + `SHIFT` + `1` / `2` / `3` / `...` panel full view
6. `ALT` + `UP` / `DOWN` move line of code
7. `CTRL` + `SHIFT` + `P` rerun previous code region
8. `CTRL` + `SHIFT` + `S` / `ENTER` source document
9. `CTRL` + `SHIFT` + `C` (un)comment lines
10. `CTRL` + `SHIFT` + `M` insert pipe operator

Rstudio Tips and Tricks

1. RStudio IDE Easy Tricks You Might've Missed and rviews.rstudio.com/categories/tips-and-tricks
2. Work with Rstudio projects!
3. `ALT` + mouse for multiline cursor. *or*: `CTRL` + `ALT` + `UP` / `DOWN`
4. `CTRL` + `1` cursor to panel (1:source, 2:console, 3:help, 4:history, 5:files, 6:plot...)
5. `CTRL` + `SHIFT` + `1` / `2` / `3` / `...` panel full view
6. `ALT` + `UP` / `DOWN` move line of code
7. `CTRL` + `SHIFT` + `P` rerun previous code region
8. `CTRL` + `SHIFT` + `S` / `ENTER` source document
9. `CTRL` + `SHIFT` + `C` (un)comment lines
10. `CTRL` + `SHIFT` + `M` insert pipe operator
11. tearable panes

Rstudio Tips and Tricks

1. RStudio IDE Easy Tricks You Might've Missed and rviews.rstudio.com/categories/tips-and-tricks
2. Work with Rstudio projects!
3. `ALT` + mouse for multiline cursor. *or*: `CTRL` + `ALT` + `UP` / `DOWN`
4. `CTRL` + `1` cursor to panel (1:source, 2:console, 3:help, 4:history, 5:files, 6:plot...)
5. `CTRL` + `SHIFT` + `1` / `2` / `3` / `...` panel full view
6. `ALT` + `UP` / `DOWN` move line of code
7. `CTRL` + `SHIFT` + `P` rerun previous code region
8. `CTRL` + `SHIFT` + `S` / `ENTER` source document
9. `CTRL` + `SHIFT` + `C` (un)comment lines
10. `CTRL` + `SHIFT` + `M` insert pipe operator
11. tearable panes
12. `CTRL` + `SHIFT` + `O` Document outline

Getting help

```
help("apply") # get help for the function
?apply # The same. saves typing effort.
# Press F1 while the cursor is at a command.

apropos("apply")
help.search("apply")
??apply

help.start() # html help Manuals, Material, search engine
```

Getting help

```
help("apply") # get help for the function
?apply # The same. saves typing effort.
# Press F1 while the cursor is at a command.

apropos("apply")
help.search("apply")
??apply

help.start() # html help Manuals, Material, search engine
```

- StackOverflow for programming questions <– main resource

Getting help

```
help("apply") # get help for the function
?apply # The same. saves typing effort.
# Press F1 while the cursor is at a command.

apropos("apply")
help.search("apply")
??apply

help.start() # html help Manuals, Material, search engine
```

- StackOverflow for programming questions <– main resource
- (online) Book: Grolemund & Wickham (2017) - R for Data Science

Getting help

```
help("apply") # get help for the function
?apply # The same. saves typing effort.
# Press F1 while the cursor is at a command.

apropos("apply")
help.search("apply")
??apply

help.start() # html help Manuals, Material, search engine
```

- StackOverflow for programming questions <– main resource
- (online) Book: Grolemund & Wickham (2017) - R for Data Science
- Reference Card by Tom Short & Jonas Stein

Getting help

```
help("apply") # get help for the function
?apply # The same. saves typing effort.
# Press F1 while the cursor is at a command.

apropos("apply")
help.search("apply")
??apply

help.start() # html help Manuals, Material, search engine
```

- StackOverflow for programming questions <– main resource
- (online) Book: Grolemund & Wickham (2017) - R for Data Science
- Reference Card by Tom Short & Jonas Stein
- More at bookdown.org/brry/course/resources

Outline

R course info
1. One-Session-Intro
2. Objects
   1. Rstudio + getting help
   2. R syntax
   3. Assignments
   4. Vectors, indexing
   5. Data.frames
   6. Matrices, apply function
   7. Lists, lapply/sapply functions
   8. Classes and methods, data types and object types
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Please calculate the following tasks

Exercise 8: Basic operators

1. 4 * 9
2. 2 + 4,8 (International decimal sign: ".")
3. 3^2
4. $\frac{\pi}{2}$
5. sinus of 15 °. Must be 0.26. Remember: $radians = degrees * \frac{\pi}{180\,^\circ}$
6. $\sqrt{81}$ (square root)
7. $|-12|$ (absolute value)
8. log(100) (–> should return 2, as $10^2 = 100$)
9. 5! (factorial)
10. e^3 (exponential function)
11. Bonus: How is 3.91 * 10^{-3} written in scientific notation?

Solution for exercise 8.1-8: Basic operators

```
4*9 # this is our first calculation with a comment
2 + 4.8
3^2
pi/2
sin(15*pi/180)
?sqrt # get the offline help about the function sqrt
sqrt(81) # square root
abs(-12) # absolute value
log(100)
help(log) # natural logarithm
log10(100) # log base 10
log(100, base=10) # another way
```

Solution for exercise 8.9-11: Basic operators

```
factorial(5)
e^3 # does not work
??exponential
exp(3) # exponential function
exp(1) # the value of e
3.91 * 10^-3
3.91e-3 # no spaces possible in scientific notation
help.start() # offline search, manuals, etc
```

Source code of R commands

Source code: github.com/wch/r-source/src/library/base (or stats, utils, ...)
CRAN packages source code: github.com/cran
HTML Documentation: www.rdocumentation.org

excellent post
ncol - .Primitive("dim") - NROW
head.matrix
`https://github.com/wch/r-source/blob/trunk/src/library/utils/R/head.R`
`https://github.com/wch/r-source/blob/trunk/src/library/base/R/matrix.R`

github.com/brry/rskey#funSource

Outline

R course info
1. One-Session-Intro
2. Objects
   1. Rstudio + getting help
   2. R syntax
   3. Assignments
   4. Vectors, indexing
   5. Data.frames
   6. Matrices, apply function
   7. Lists, lapply/sapply functions
   8. Classes and methods, data types and object types
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Assignment: Create an object with data in the workspace I

```
a <- 15.4
```

Assignment: Create an object with data in the workspace I

```
a <- 15.4
```

```
a

## [1] 15.4
```

Assignment: Create an object with data in the workspace I

```
a <- 15.4
```

```
a

## [1] 15.4
```

```
a > 2 # is a bigger than 2?  --> logical value (boolean)

## [1] TRUE
```

Assignment: Create an object with data in the workspace I

```
a <- 15.4
```

```
a

## [1] 15.4
```

```
a > 2 # is a bigger than 2?  --> logical value (boolean)

## [1] TRUE
```

```
A # not an existing object (R is case sensitive)

## Error in eval(expr, envir, enclos):  object 'A' not found
```

Assignment: Create an object with data in the workspace II

Don't assign to existing objects or functions like `mean, c, T, data, sin`; Later on, you will hardly be able to distinguish what object (original or custom) you are referring to.

Assignment: Create an object with data in the workspace II

Don't assign to existing objects or functions like `mean, c, T, data, sin`; Later on, you will hardly be able to distinguish what object (original or custom) you are referring to.
As R is Case SeNsitivE, `Data` would be OK, but it's not useful. Use object names that are short, but still meaningful.

Assignment: Create an object with data in the workspace II

Don't assign to existing objects or functions like `mean, c, T, data, sin`; Later on, you will hardly be able to distinguish what object (original or custom) you are referring to.
As R is Case SeNsitivE, `Data` would be OK, but it's not useful. Use object names that are short, but still meaningful.
One convention is lowerCamel:
`dailyRainBerlin` is short and readable through capital letters.

Assignment: Create an object with data in the workspace II

Don't assign to existing objects or functions like `mean, c, T, data, sin`; Later on, you will hardly be able to distinguish what object (original or custom) you are referring to.
As R is Case SeNsitivE, `Data` would be OK, but it's not useful. Use object names that are short, but still meaningful.
One convention is lowerCamel:
`dailyRainBerlin` is short and readable through capital letters.

```
# Rstudio: press 'ALT' + '-'  for   " <- "
```

Assignment: Create an object with data in the workspace II

Don't assign to existing objects or functions like `mean, c, T, data, sin`; Later on, you will hardly be able to distinguish what object (original or custom) you are referring to.
As R is Case SeNsitivE, `Data` would be OK, but it's not useful. Use object names that are short, but still meaningful.
One convention is lowerCamel:
`dailyRainBerlin` is short and readable through capital letters.

```
# Rstudio: press 'ALT' + '-'  for   " <- "
```

Don't use `=` for assignments. Most R users connotate "=" with function arguments and prefer " <- " for objects (style guide). Also, "=" is only allowed on the top level:

Assignment: Create an object with data in the workspace II

Don't assign to existing objects or functions like `mean, c, T, data, sin`; Later on, you will hardly be able to distinguish what object (original or custom) you are referring to.
As R is Case SeNsitivE, `Data` would be OK, but it's not useful. Use object names that are short, but still meaningful.
One convention is lowerCamel:
`dailyRainBerlin` is short and readable through capital letters.

```
# Rstudio: press 'ALT' + '-'  for   " <- "
```

Don't use `=` for assignments. Most R users connotate "=" with function arguments and prefer " <- " for objects (style guide). Also, "=" is only allowed on the top level:
`median(x <- 1:10)` also creates the object in `globalenv()`, while `median(x=1:10)` does not. This is used in `NCOL`, for example.

Assignment: Create an object with data in the workspace II

Don't assign to existing objects or functions like `mean, c, T, data, sin`; Later on, you will hardly be able to distinguish what object (original or custom) you are referring to.
As R is Case SeNsitivE, `Data` would be OK, but it's not useful. Use object names that are short, but still meaningful.
One convention is lowerCamel:
`dailyRainBerlin` is short and readable through capital letters.

```
# Rstudio: press 'ALT' + '-'  for   " <- "
```

Don't use `=` for assignments. Most R users connotate "=" with function arguments and prefer " <- " for objects (style guide). Also, "=" is only allowed on the top level:
`median(x <- 1:10)` also creates the object in `globalenv()`, while `median(x=1:10)` does not. This is used in `NCOL`, for example.

http://blog.revolutionanalytics.com/2008/12/use-equals-or-arrow-for-assignment.html

https://csgillespie.wordpress.com/2010/11/16/assignment-operators-in-r-vs/

Outline

R course info
1. One-Session-Intro
2. Objects
   1. Rstudio + getting help
   2. R syntax
   3. Assignments
   4. Vectors, indexing
   5. Data.frames
   6. Matrices, apply function
   7. Lists, lapply/sapply functions
   8. Classes and methods, data types and object types
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Vectors I

In R, Vectors are not geometric constructs, but an ordered set of values. Create a vector with `c` (stands for Combine or Concatenate), separating each entry with a comma:

Vectors I

In R, Vectors are not geometric constructs, but an ordered set of values. Create a vector with c (stands for Combine or Concatenate), separating each entry with a comma:

```
b <- c(3, 7, -2.5, 11, 3.8, 9)
```

Vectors I

In R, Vectors are not geometric constructs, but an ordered set of values. Create a vector with c (stands for Combine or Concatenate), separating each entry with a comma:

```
b <- c(3, 7, -2.5, 11, 3.8, 9)
```

spaces make code easier to read:

```
b<-c(3,7,-2.5,11,3.8,9)
b <- c(3, 7, -2.5, 11, 3.8, 9)
```

Vectors I

In R, Vectors are not geometric constructs, but an ordered set of values. Create a vector with c (stands for Combine or Concatenate), separating each entry with a comma:

```
b <- c(3, 7, -2.5, 11, 3.8, 9)
```

spaces make code easier to read:

```
b<-c(3,7,-2.5,11,3.8,9)
b <- c(3, 7, -2.5, 11, 3.8, 9)
```

```
print(b) # or just write   b

## [1]  3.0  7.0 -2.5 11.0  3.8  9.0
```

Vectors II: Other ways to generate vectors

Vectors II: Other ways to generate vectors

```
1:5  # integer numbers  from : to

## [1] 1 2 3 4 5
```

Vectors II: Other ways to generate vectors

```
1:5  # integer numbers  from : to

## [1] 1 2 3 4 5
```

```
rep(1:4, times=3) # repeat entries several times

##  [1] 1 2 3 4 1 2 3 4 1 2 3 4
```

Vectors II: Other ways to generate vectors

```
1:5  # integer numbers  from : to

## [1] 1 2 3 4 5
```

```
rep(1:4, times=3) # repeat entries several times

##  [1] 1 2 3 4 1 2 3 4 1 2 3 4
```

```
v <- rep(1:3, each=3, times=2)
v

##  [1] 1 1 1 2 2 2 3 3 3 1 1 1 2 2 2 3 3 3
```

Vectors III: Yet other ways to generate vectors

Vectors III: Yet other ways to generate vectors

```
seq(from=5, to=-1, by=-0.5) # Sequence
# by must be negative for descending sequences

##  [1]  5.0  4.5  4.0  3.5  3.0  2.5  2.0  1.5  1.0
## [10]  0.5  0.0 -0.5 -1.0
```

Vectors III: Yet other ways to generate vectors

```
seq(from=5, to=-1, by=-0.5) # Sequence
# by must be negative for descending sequences

##  [1]  5.0  4.5  4.0  3.5  3.0  2.5  2.0  1.5  1.0
## [10]  0.5  0.0 -0.5 -1.0
```

```
seq(1.32, 6.1, length.out=15) # 15 elements

##  [1] 1.320000 1.661429 2.002857 2.344286 2.685714
##  [6] 3.027143 3.368571 3.710000 4.051429 4.392857
## [11] 4.734286 5.075714 5.417143 5.758571 6.100000
```

Vectors III: Yet other ways to generate vectors

```
seq(from=5, to=-1, by=-0.5) # Sequence
# by must be negative for descending sequences

##  [1]  5.0  4.5  4.0  3.5  3.0  2.5  2.0  1.5  1.0
## [10]  0.5  0.0 -0.5 -1.0
```

```
seq(1.32, 6.1, length.out=15) # 15 elements

##  [1] 1.320000 1.661429 2.002857 2.344286 2.685714
##  [6] 3.027143 3.368571 3.710000 4.051429 4.392857
## [11] 4.734286 5.075714 5.417143 5.758571 6.100000
```

```
seq(1.32, 6.1, len=15) # abbreviate argument names
?seq # read more about it...
```

Indexing on vectors (subsets) I: square brackets

```
b # remember b?

## [1]  3.0  7.0 -2.5 11.0  3.8  9.0

b[1] # first element

## [1] 3
```

Indexing on vectors (subsets) I: square brackets

```
b # remember b?

## [1]  3.0  7.0 -2.5 11.0  3.8  9.0

b[1] # first element

## [1] 3
```

```
b[2:4] # several elements

## [1]  7.0 -2.5 11.0

b[ c(2,5,1,6,1) ] # flexible

## [1] 7.0 3.8 3.0 9.0 3.0
```

Indexing on vectors (subsets) II

```
b

## [1]  3.0  7.0 -2.5 11.0  3.8  9.0

b[-2] # all but the second element

## [1]  3.0 -2.5 11.0  3.8  9.0
```

Indexing on vectors (subsets) II

```
b

## [1]  3.0  7.0 -2.5 11.0  3.8  9.0

b[-2] # all but the second element

## [1]  3.0 -2.5 11.0  3.8  9.0
```

```
a <- seq(from=1, to=100, by=0.1)

head(a) # only first 6 elements (or rows)

## [1] 1.0 1.1 1.2 1.3 1.4 1.5
```

Indexing on vectors (subsets) III

```
head(a)

## [1] 1.0 1.1 1.2 1.3 1.4 1.5

a[2] <- 87 # manipulate single elements of object
head(a)

## [1]  1.0 87.0  1.2  1.3  1.4  1.5
```

```
ls() # list objects in workspace

## [1] "a" "b" "v"

rm(b) # remove an object
```

Data types

Data types

R can handle data of a variety of types:

Data types

R can handle data of a variety of types: Numbers like `42.6`,

Data types

R can handle data of a variety of types: Numbers like `42.6`, characters like `"R rocks"`,

Data types

R can handle data of a variety of types: Numbers like `42.6`, characters like `"R rocks"`, complex numbers like `8+4i`,

Data types

R can handle data of a variety of types: Numbers like `42.6`, characters like `"R rocks"`, complex numbers like `8+4i`, categories with `as.factor(c("blue", "green"))`,

Data types

R can handle data of a variety of types: Numbers like `42.6`, characters like `"R rocks"`, complex numbers like `8+4i`, categories with `as.factor(c("blue", "green"))`, logical or boolean values `c(TRUE, FALSE)`

Data types

R can handle data of a variety of types: Numbers like `42.6`, characters like `"R rocks"`, complex numbers like `8+4i`, categories with `as.factor(c("blue", "green"))`, logical or boolean values `c(TRUE, FALSE)` and a few others.

Data types

R can handle data of a variety of types: Numbers like `42.6`, characters like `"R rocks"`, complex numbers like `8+4i`, categories with `as.factor(c("blue", "green"))`, logical or boolean values `c(TRUE, FALSE)` and a few others. They should usually not be mixed.

Data types

R can handle data of a variety of types: Numbers like `42.6`, characters like `"R rocks"`, complex numbers like `8+4i`, categories with `as.factor(c("blue", "green"))`, logical or boolean values `c(TRUE, FALSE)` and a few others. They should usually not be mixed. For example, you can't perform algebraic computation on characters:

Data types

R can handle data of a variety of types: Numbers like `42.6`, characters like `"R rocks"`, complex numbers like `8+4i`, categories with `as.factor(c("blue", "green"))`, logical or boolean values `c(TRUE, FALSE)` and a few others. They should usually not be mixed. For example, you can't perform algebraic computation on characters:

```
p9 <- "1.3"  ; p9 # show assignment result
```

Data types

R can handle data of a variety of types: Numbers like `42.6`, characters like `"R rocks"`, complex numbers like `8+4i`, categories with `as.factor(c("blue", "green"))`, logical or boolean values `c(TRUE, FALSE)` and a few others. They should usually not be mixed. For example, you can't perform algebraic computation on characters:

```
p9 <- "1.3"  ; p9 # show assignment result
```

```
2*p9              # error, as p9 is a character
```

Data types

R can handle data of a variety of types: Numbers like `42.6`, characters like `"R rocks"`, complex numbers like `8+4i`, categories with `as.factor(c("blue", "green"))`, logical or boolean values `c(TRUE, FALSE)` and a few others. They should usually not be mixed. For example, you can't perform algebraic computation on characters:

```
p9 <- "1.3"  ; p9 # show assignment result
2*p9              # error, as p9 is a character
2*as.numeric(p9)  # make it a number first
```

Data types

R can handle data of a variety of types: Numbers like `42.6`, characters like `"R rocks"`, complex numbers like `8+4i`, categories with `as.factor(c("blue", "green"))`, logical or boolean values `c(TRUE, FALSE)` and a few others. They should usually not be mixed. For example, you can't perform algebraic computation on characters:

```
p9 <- "1.3"  ; p9 # show assignment result
2*p9              # error, as p9 is a character
2*as.numeric(p9)  # make it a number first
p9                # p9 itself has not changed
```

Data types

R can handle data of a variety of types: Numbers like `42.6`, characters like `"R rocks"`, complex numbers like `8+4i`, categories with `as.factor(c("blue", "green"))`, logical or boolean values `c(TRUE, FALSE)` and a few others. They should usually not be mixed. For example, you can't perform algebraic computation on characters:

```
p9 <- "1.3"  ; p9 # show assignment result
2*p9              # error, as p9 is a character
2*as.numeric(p9)  # make it a number first
p9                # p9 itself has not changed
is.numeric(p9)    # FALSE
```

Data types

R can handle data of a variety of types: Numbers like `42.6`, characters like `"R rocks"`, complex numbers like `8+4i`, categories with `as.factor(c("blue", "green"))`, logical or boolean values `c(TRUE, FALSE)` and a few others. They should usually not be mixed. For example, you can't perform algebraic computation on characters:

```
p9 <- "1.3"  ; p9 # show assignment result
2*p9              # error, as p9 is a character
2*as.numeric(p9)  # make it a number first
p9                # p9 itself has not changed
is.numeric(p9)    # FALSE
mode(p9)          # character
```

order of coercion

Integer vs decimal numbers: arithmetic precision

```
6  ==  3 + 3            # TRUE
```

Integer vs decimal numbers: arithmetic precision

```
6  ==  3 + 3            # TRUE
```

```
0.5 - 0.2  ==  0.3      # TRUE
```

Integer vs decimal numbers: arithmetic precision

```
6  ==  3 + 3          # TRUE
```

```
0.5 - 0.2  ==  0.3    # TRUE
```

```
0.4 - 0.1  ==  0.3    # FALSE
```

Integer vs decimal numbers: arithmetic precision

```
6  ==  3 + 3            # TRUE
```

```
0.5 - 0.2  ==  0.3      # TRUE
```

```
0.4 - 0.1  ==  0.3      # FALSE
```

```
print(0.4-0.1 , digits=22)

## [1] 0.30000000000000004
```

Integer vs decimal numbers: arithmetic precision

```
6  ==  3 + 3            # TRUE
```

```
0.5 - 0.2  ==  0.3      # TRUE
```

```
0.4 - 0.1  ==  0.3      # FALSE
```

```
print(0.4-0.1 , digits=22)

## [1] 0.30000000000000004
```

```
round(0.4-0.1, digits=6)  ==  round(0.3, 6)

## [1] TRUE
```

Integer vs decimal numbers: arithmetic precision

```
6  ==  3 + 3            # TRUE
```

```
0.5 - 0.2  ==  0.3      # TRUE
```

```
0.4 - 0.1  ==  0.3      # FALSE
```

```
print(0.4-0.1 , digits=22)

## [1] 0.30000000000000004
```

```
round(0.4-0.1, digits=6)  ==  round(0.3, 6)

## [1] TRUE
```

Test for equality only with integers / with rounding (see also `signif`), or use `all.equal`.

Integer vs decimal numbers: arithmetic precision

```
6  ==  3 + 3            # TRUE
```

```
0.5 - 0.2  ==  0.3      # TRUE
```

```
0.4 - 0.1  ==  0.3      # FALSE
```

```
print(0.4-0.1 , digits=22)

## [1] 0.30000000000000004
```

```
round(0.4-0.1, digits=6)  ==  round(0.3, 6)

## [1] TRUE
```

Test for equality only with integers / with rounding (see also `signif`), or use `all.equal`. More pitfalls like this in the R inferno.

Exercise 9: Vectors and statistics

1. Repeat your favorite number 6 times using `rep`.
2. With `sequence`, generate one vector from -1 to 4 in steps of 0.5 and one with a length of 21 elements. Then add a number to the vector.
3. Assign a vector with the body heights of a few people around you to an object with a useful name.
4. Print only the 4th and 2nd element (one single vector). BONUS: All but the 2nd to 4th elements.
5. `sort` the numbers decreasingly.
6. Calculate the average size (`mean`), the variance (`var`) and the standard deviation (`sd`) of this dataset, as well as the `min`, `max`, `median` and 80% `quantile` (20% of the entries are larger than this value).
7. BONUS 1: Explain what `length` does.
8. BONUS 2: Generate the numbers from 1 to 10 in three different ways.
9. BONUS 3: Explain and relate to each other: mean, var, sd, median, quantile.

Solution for exercise 9.1-2: Vectors

```
# repeat and sequence:
rep(5, times=6)
seq(from=-1, to=4, by=0.5)
vec2 <- seq(-1, 4, length.out=21)
seq(-1, 4, len=21)
```

You can leave argument names away, if you specify their content in the right order. If it is unambiguous, you can use partial argument names.

```
# add a number to a vector:
c(vec2, 8) # does not change vec2
vec3 <- c(vec2, 8) # creates a new object
vec2 <- c(vec2, 8) # changes the object vec2
# Another way:
vec2 <- seq(-1, 4, length.out=21)
vec2[21] <- 5 # changing one element
vec2
vec2[22] # returns NA - there is no value here
vec2[26] <- 8
vec2 # is now changed
```

Solution for exercise 9.3-6: Vectors

```
# vector with body heights:
heights <- c(192, 171, 178, 175, 185, 145, 164) # cm
# print selected elements:
heights[ c(4,2) ]
# sort:
sort(heights, decreasing=T)
# statistics:
mean(heights)
var(heights) # cm^2
sd(heights) # cm
min(heights)
max(heights)
median(heights)
quantile(heights, probs=0.80)
```

Solution for exercise 9.BONUS: Vectors

```
length(heights) # number of elements in vector
# Numbers from 1 to 10:
1:10  ;  seq(1,10)  ;  c(1,2,3,4,5,6,7,8,9,10)
# mean vs median:
mean(   replace(heights, 1, 292) ) # subject to outliers
median( replace(heights, 1, 292) ) # indifferent to extrema
```

70% of values in a normal distribution lie between mean-sd and mean+sd:

Outline

R course info
1. One-Session-Intro
2. Objects
   1. Rstudio + getting help
   2. R syntax
   3. Assignments
   4. Vectors, indexing
   5. Data.frames
   6. Matrices, apply function
   7. Lists, lapply/sapply functions
   8. Classes and methods, data types and object types
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

data.frames

`data.frame`s are the most common form of tables in R.

data.frames

`data.frame`s are the most common form of tables in R. Each column can have a different data type (numeric, character, factor, etc), but within a column, it stays the same.

data.frames

`data.frame`s are the most common form of tables in R. Each column can have a different data type (numeric, character, factor, etc), but within a column, it stays the same.
Columns as well as rows can have names (`colnames, rownames`). If these start with a number, an "X" will be prefixed.

data.frames

`data.frame`s are the most common form of tables in R. Each column can have a different data type (numeric, character, factor, etc), but within a column, it stays the same.
Columns as well as rows can have names (`colnames, rownames`). If these start with a number, an "X" will be prefixed. This may happen when reading data, which is usually stored as a `data.frame` in R.

data.frames

`data.frame`s are the most common form of tables in R. Each column can have a different data type (numeric, character, factor, etc), but within a column, it stays the same.
Columns as well as rows can have names (`colnames, rownames`). If these start with a number, an "X" will be prefixed. This may happen when reading data, which is usually stored as a `data.frame` in R.

```
?data.frame
```

Creating data.frames

```
df <- data.frame(col_a=11:14, ColName2=letters[1:4],
                 LogicalCol=(1:4)>2)
df

##   col_a ColName2 LogicalCol
## 1    11        a      FALSE
## 2    12        b      FALSE
## 3    13        c       TRUE
## 4    14        d       TRUE

str(df)

## 'data.frame': 4 obs. of  3 variables:
##  $ col_a     : int  11 12 13 14
##  $ ColName2  : Factor w/ 4 levels "a","b","c","d": 1 2 3 4
##  $ LogicalCol: logi  FALSE FALSE TRUE TRUE
```

Subsetting data.frames by index numbers

```
nrow(df)

## [1] 4

ncol(df) # see also dim(df)

## [1] 3
```

Subsetting data.frames by index numbers

```r
nrow(df)

## [1] 4

ncol(df) # see also dim(df)

## [1] 3
```

```r
df[ 3 , 1] # 3rd row, 1st column

## [1] 13

df[   , 2] # all rows, 2nd column (as vector)

## [1] a b c d
## Levels: a b c d
```

Subsetting data.frames by names

```
df[ "LogicalCol" ] # no comma -> data.frame

##   LogicalCol
## 1      FALSE
## 2      FALSE
## 3       TRUE
## 4       TRUE
```

Bad practice, unclear to user whether rows or columns are indexed!

```
df[ , "LogicalCol", drop=FALSE ]  # better
```

Subsetting data.frames by names

```
df[ "LogicalCol" ] # no comma -> data.frame

##   LogicalCol
## 1      FALSE
## 2      FALSE
## 3       TRUE
## 4       TRUE
```

Bad practice, unclear to user whether rows or columns are indexed!

```
df[ , "LogicalCol", drop=FALSE ]  # better
```

```
df$col_a

## [1] 11 12 13 14
```

Subsetting data.frames by names

```
df[ "LogicalCol" ] # no comma -> data.frame

##   LogicalCol
## 1      FALSE
## 2      FALSE
## 3       TRUE
## 4       TRUE
```

Bad practice, unclear to user whether rows or columns are indexed!

```
df[ , "LogicalCol", drop=FALSE ]  # better
```

```
df$col_a

## [1] 11 12 13 14
```

```
df[2, ] # full row

##   col_a ColName2 LogicalCol
## 2    12        b      FALSE
```

Some hints about data.frames

- If `nrow(df)` returns `NULL`, `df` might in fact be a vector.

Some hints about data.frames

- If `nrow(df)` returns `NULL`, `df` might in fact be a vector.
- `NROW(df)` will show `length(df)` if `df` is a vector.

Some hints about data.frames

- If `nrow(df)` returns `NULL`, `df` might in fact be a vector.
- `NROW(df)` will show `length(df)` if `df` is a vector.
- Also, remember you can check the object with `class` and `mode`, as well as `is.vector`.

Some hints about data.frames

- If `nrow(df)` returns `NULL`, `df` might in fact be a vector.
- `NROW(df)` will show `length(df)` if `df` is a vector.
- Also, remember you can check the object with `class` and `mode`, as well as `is.vector`.
- Objects like matrices can be converted with `as.data.frame(theMatrix)`

Some hints about data.frames

- If `nrow(df)` returns `NULL`, `df` might in fact be a vector.
- `NROW(df)` will show `length(df)` if `df` is a vector.
- Also, remember you can check the object with `class` and `mode`, as well as `is.vector`.
- Objects like matrices can be converted with `as.data.frame(theMatrix)`
- `colnames(df)` will show the column names. (see also `rownames`)

Exercise 10: data.frames

Working with the dataset `state.x77` (comes built-in with R)

1. Change the object type to a data.frame.
2. What are the 3 ways to select the population column? Which one doesn't work for Life expectancy? Why not? (This only applies if you used `as.data.frame`, not `data.frame`)
3. How can you subset a single column, without the result being simplified to a vector? (you want to keep it as a data.frame, but with one single column only. Hint: `?"["`)
4. How could the `$` subsetting method be dangerous when misspelling columns?
5. By which two ways can you select the Income and Illiteracy columns?
6. How can you select all the states (=rows) with an income above 5000?
7. BONUS: Why does `state.x77$Income` not work on the original dataset? (use `rm(state.x77)` to remove your changed object from the workspace, if you didn't assign it to a different name.)

Solution to exercise 10: subsetting data.frames

```
state <- as.data.frame(state.x77)
state[,1] # not safe, columns could be in different position
state[ , "Population"] # safe and readable
state$Population # quicker to type, has autocomplete
# $  doesn't work (at least not without quotation marks)
   # with a space, it's an invalid column name
state[ , "Population", drop=FALSE]
state$Illeeteeraacyy # does not give an error!!!
state[ , c("Income","Illiteracy")]
state[ , 2:3]
state[state$Income>=5000, ]
# Because it is a matrix, not a data.frame
```

Outline

R course info
1. One-Session-Intro
2. Objects
   1. Rstudio + getting help
   2. R syntax
   3. Assignments
   4. Vectors, indexing
   5. Data.frames
   6. Matrices, apply function
   7. Lists, lapply/sapply functions
   8. Classes and methods, data types and object types
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Creating matrices

```r
matrix(data=1:6 , nrow=2, ncol=3)

##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Creating matrices

```
matrix(data=1:6 , nrow=2, ncol=3)

##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
m <- matrix(1:6 , nrow=2, ncol=3, byrow=T)
m

##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

Manipulating matrices, row and column names

```
colnames(m) <- c("A", "B", "C")
m[1,1] <- c(989)
m

##         A B C
## [1,] 989 2 3
## [2,]   4 5 6
```

Manipulating matrices, row and column names

```
colnames(m) <- c("A", "B", "C")
m[1,1] <- c(989)
m

##        A B C
## [1,] 989 2 3
## [2,]   4 5 6
```

```
rownames(m) <- c("row1", "row2")
m

##        A B C
## row1 989 2 3
## row2   4 5 6
```

Matrices only have one single data type

```
m[1,1] <- "a"
m

##      A   B   C
## row1 "a" "2" "3"
## row2 "4" "5" "6"
```

All are converted to character (see order of coercion)

Apply function to matrix rows / columns

```
m <- matrix(1:12, ncol=4)
rowMeans(m)

## [1] 5.5 6.5 7.5

apply(m, MARGIN=1, FUN=median) # MARGIN=1: Keep x dimension

## [1] 5.5 6.5 7.5

apply(m, MARGIN=2, median, na.rm=TRUE) # apply to columns

## [1]  2  5  8 11
```

combining matrices

```
m <- matrix(11:16, ncol=3) ; m # 2 x 3

##      [,1] [,2] [,3]
## [1,]   11   13   15
## [2,]   12   14   16

p <- matrix(21:32, ncol=4) ; p # 3 x 4

##      [,1] [,2] [,3] [,4]
## [1,]   21   24   27   30
## [2,]   22   25   28   31
## [3,]   23   26   29   32

r <- matrix(31:39, ncol=3) ; r # 3 x 3

##      [,1] [,2] [,3]
## [1,]   31   34   37
## [2,]   32   35   38
## [3,]   33   36   39
```

combining matrices m: 2 x 3 p: 3 x 4 r: 3 x 3

```
rbind(m,r) # row-bind: number of columns must match

##      [,1] [,2] [,3]
## [1,]   11   13   15
## [2,]   12   14   16
## [3,]   31   34   37
## [4,]   32   35   38
## [5,]   33   36   39

# rbind(m,p) # Error: 3 and 4 columns
```

combining matrices m: 2 x 3 p: 3 x 4 r: 3 x 3

```
rbind(m,r) # row-bind: number of columns must match

##      [,1] [,2] [,3]
## [1,]   11   13   15
## [2,]   12   14   16
## [3,]   31   34   37
## [4,]   32   35   38
## [5,]   33   36   39

# rbind(m,p) # Error: 3 and 4 columns
```

```
cbind(p,r) # column-bind: nrows must match

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## [1,]   21   24   27   30   31   34   37
## [2,]   22   25   28   31   32   35   38
## [3,]   23   26   29   32   33   36   39

# cbind(m,r) # Error: 2 and 3 rows3
```

combining matrices

```
m

##      [,1] [,2] [,3]
## [1,]   11   13   15
## [2,]   12   14   16

t(m) # transpose

##      [,1] [,2]
## [1,]   11   12
## [2,]   13   14
## [3,]   15   16
```

Write a function and apply it to a matrix

Exercise 11: functions + matrices

1. What do you get with `vect <- c(5, 3, 9, 1, 7, 4, 8, 6) ; which.min(vect)`?
2. How do you exclude the sixt element of `vect` (return all of `vect` except the value at position 6)?
3. Write a function to get the second-largest value of a vector and test it on `vect`. Hint: figure out a way to exclude the element which is max. Task 1+2 might help.
4. Create a matrix with the integers from 1 to 12 organized in three columns (filled column-wise).
5. Apply the function row-wise and column-wise.
6. What type of object does that return? (Hint: it's not a matrix as you might have expected.)
7. BONUS: Write a function that can optionally return the third/fourth/etc-largest value of a vector. Remember to use curly braces (`{...}`) around the body of the function if several statements are involved. You can create and access objects within a function in the same way you would in the global workspace.
8. tough BONUS: Have the function `stop` with an informative error if the number to exclude is larger than the length of the vector. Handle the input 0.

Solution for exercise 11.1-3: functions + matrices

`which.min` returns the position (index) of smallest value.

```
vect <- c(5, 3, 9, 1, 7, 4, 8, 6)
secondMax <- function(x) max(x[-which.max(x)])
secondMax(vect)

## [1] 8
```

Solution for exercise 11.4-6: functions + matrices

```
z <- matrix(1:12, ncol=3)   ;   z

##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12

apply(z, MARGIN=1, secondMax)

## [1] 5 6 7 8

apply(z, MARGIN=2, secondMax)

## [1]  3  7 11
```

MARGIN: dimension to keep when function reduces vectors to a single number. apply then takes these single numbers (one per row/column) and combines them into a vector.

Solution for exercise 11.7-8: functions + matrices

```
nthMax <- function(x, n_exclude=0)
  {
  x <- sort(x, decreasing=TRUE)
  if(n_exclude >= length(x))
     stop("Can't exclude ", n_exclude,
          " elements from a vector with length ", length(x))
  if(n_exclude > 0) x <- x[-(1:n_exclude)]
  max(x)
}
nthMax(vect) ; max(vect)              # 9
nthMax(vect, 1) ; secondMax(vect)     # 8
nthMax(vect, 2)                       # 7
```

Outline

R course info
1. One-Session-Intro
2. Objects
   1. Rstudio + getting help
   2. R syntax
   3. Assignments
   4. Vectors, indexing
   5. Data.frames
   6. Matrices, apply function
   7. Lists, lapply/sapply functions
   8. Classes and methods, data types and object types
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Lists

Lists store any R object type (vectors, data.frames, . . .) in one object.

```
list("df"=df[1:2, ], matrix=m, character="TEXT")

## $df
##   col_a ColName2 LogicalCol
## 1    11        a      FALSE
## 2    12        b      FALSE
##
## $matrix
##      [,1] [,2] [,3]
## [1,]   11   13   15
## [2,]   12   14   16
##
## $character
## [1] "TEXT"
```

List basics I

```
# create a list: an object with several 'objects' inside it
MYLIST <- list(first="result of function 'list'",
               second=345:287, third=rnorm(10),
               fourth=data.frame(a=6:3, b=rexp(4)),
               fifth=LETTERS, sixth=letters,
               seventh=sample(300), eigth="goodbye")
```

List basics I

```
# create a list: an object with several 'objects' inside it
MYLIST <- list(first="result of function 'list'",
               second=345:287, third=rnorm(10),
               fourth=data.frame(a=6:3, b=rexp(4)),
               fifth=LETTERS, sixth=letters,
               seventh=sample(300), eigth="goodbye")
```

```
# print the whole thing:
MYLIST
```

List basics I

```
# create a list: an object with several 'objects' inside it
MYLIST <- list(first="result of function 'list'",
               second=345:287, third=rnorm(10),
               fourth=data.frame(a=6:3, b=rexp(4)),
               fifth=LETTERS, sixth=letters,
               seventh=sample(300), eigth="goodbye")
```

```
# print the whole thing:
MYLIST
```

```
# print only the first 4 elements of the list
head(MYLIST,4)
tail(MYLIST) # or the last n elements (by default 6)
```

List basics II

```r
# Structure of the list
str(MYLIST, max.level=1, vec.len=3)

## List of 8
##  $ first  : chr "result of function 'list'"
##  $ second : int [1:59] 345 344 343 342 341 340 339 338 ...
##  $ third  : num [1:10] 1.076 -0.864 0.0625 0.1375 ...
##  $ fourth :'data.frame': 4 obs. of  2 variables:
##  $ fifth  : chr [1:26] "A" "B" "C" ...
##  $ sixth  : chr [1:26] "a" "b" "c" ...
##  $ seventh: int [1:300] 6 143 214 22 114 96 223 234 ...
##  $ eigth  : chr "goodbye"
```

List basics II

```r
# str down to each level of the list
str(MYLIST, vec.len=3) # see the df for the difference

## List of 8
##  $ first  : chr "result of function 'list'"
##  $ second : int [1:59] 345 344 343 342 341 340 339 338 ...
##  $ third  : num [1:10] 1.076 -0.864 0.0625 0.1375 ...
##  $ fourth :'data.frame': 4 obs. of  2 variables:
##   ..$ a: int [1:4] 6 5 4 3
##   ..$ b: num [1:4] 0.288 1.6645 0.0847 1.6081
##  $ fifth  : chr [1:26] "A" "B" "C" ...
##  $ sixth  : chr [1:26] "a" "b" "c" ...
##  $ seventh: int [1:300] 6 143 214 22 114 96 223 234 ...
##  $ eigth  : chr "goodbye"
```

List indexing I

```
# Assessing list elements (indexing) with double square brackets:
MYLIST[[3]]

##  [1]  1.07597085 -0.86401751  0.06248087
##  [4]  0.13752332  0.01085314  0.75895050
##  [7] -1.11800498  0.35239352 -0.98079994
## [10] -1.53383837
```

List indexing I

```
# Assessing list elements (indexing) with double square brackets:
MYLIST[[3]]

##  [1]  1.07597085 -0.86401751  0.06248087
##  [4]  0.13752332  0.01085314  0.75895050
##  [7] -1.11800498  0.35239352 -0.98079994
## [10] -1.53383837
```

```
# What's different if you use only single brackets?
MYLIST[3]

## $third
##  [1]  1.07597085 -0.86401751  0.06248087
##  [4]  0.13752332  0.01085314  0.75895050
##  [7] -1.11800498  0.35239352 -0.98079994
## [10] -1.53383837

# HINT: look at the class of the results
```

List indexing II

```
class(MYLIST[3]) # list
class(MYLIST[[3]]) # numeric
```

`MYLIST[3]` returns a list (with a single element).
`MYLIST[[3]]` returns that single element as is (in this case: a vector with 10 numbers).

List indexing II

```r
class(MYLIST[3]) # list
class(MYLIST[[3]]) # numeric
```

`MYLIST[3]` returns a list (with a single element).
`MYLIST[[3]]` returns that single element as is (in this case: a vector with 10 numbers).

```r
# Several elements of the list (the result is still a list):
MYLIST[4:5]
```

```
## $fourth
##   a         b
## 1 6 0.2880489
## 2 5 1.6644699
## 3 4 0.0847095
## 4 3 1.6080530
##
## $fifth
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K"
## [12] "L" "M" "N" "O" "P" "Q" "R" "S" "T" "U" "V"
## [23] "W" "X" "Y" "Z"
```

List indexing III

```
# Assesing elements by name
# NULL if not existent, so watch out with spelling!
MYLIST$eight

## NULL

MYLIST$eigth

## [1] "goodbye"
```

List indexing III

```
# Assesing elements by name
# NULL if not existent, so watch out with spelling!
MYLIST$eight

## NULL

MYLIST$eigth

## [1] "goodbye"
```

```
names(MYLIST) # as with names(VECTOR) or rownames(DATAFRAME)

## [1] "first"   "second"  "third"   "fourth"
## [5] "fifth"   "sixth"   "seventh" "eigth"

names(MYLIST)[2] <- "second elem."
```

List indexing IV

```
# changing elements:
MYLIST[c(3,6)]  <- c(77777777, 3333333)
head(MYLIST)
```

L-apply

```
typeof(MYLIST[[1]])

## [1] "character"
```

L-apply

```
typeof(MYLIST[[1]])

## [1] "character"
```

```
# Apply a function to each element of the list:
lapply(X=MYLIST, FUN=typeof)
```

L-apply

```r
typeof(MYLIST[[1]])

## [1] "character"
```

```r
# Apply a function to each element of the list:
lapply(X=MYLIST, FUN=typeof)
```

```
## $first
## [1] "character"
##
## $`second elem.`
## [1] "integer"
##
## $third
## [1] "double"
##
## $fourth
## [1] "list"
## $fifth
## [1] "character"
##
## $sixth
## [1] "character"
##
## $seventh
## [1] "integer"
##
## $eigth
## [1] "character"
```

S-apply I

```r
# simplify that output (if possible):
    sapply(X=MYLIST, FUN=typeof)

##        first second elem.        third       fourth
##  "character"    "integer"     "double"       "list"
##        fifth        sixth      seventh        eigth
##  "character"  "character"    "integer"  "character"
```

S-apply I

```
# simplify that output (if possible):
    sapply(X=MYLIST, FUN=typeof)

##        first second elem.        third       fourth
##  "character"    "integer"     "double"       "list"
##        fifth        sixth      seventh        eigth
##  "character"  "character"    "integer"  "character"
```

```
# simplifying not possible if FUN(MYLIST[[1]]) and
# FUN(MYLIST[[2]]) return different results:
sapply(X=MYLIST, FUN=head)
```

S-apply II

```
# List with vectors:
L2 <- list(3:7, 8:4, 5:9)
L2

## [[1]]
## [1] 3 4 5 6 7
##
## [[2]]
## [1] 8 7 6 5 4
##
## [[3]]
## [1] 5 6 7 8 9
```

S-apply II

```r
# List with vectors:
L2 <- list(3:7, 8:4, 5:9)
L2

## [[1]]
## [1] 3 4 5 6 7
##
## [[2]]
## [1] 8 7 6 5 4
##
## [[3]]
## [1] 5 6 7 8 9
```

```r
sapply(L2, I)

##      [,1] [,2] [,3]
## [1,]    3    8    5
## [2,]    4    7    6
## [3,]    5    6    7
## [4,]    6    5    8
## [5,]    7    4    9
```

`I` basically returns the input as is. This is useful for `tapply` results!

S-apply III

```
# That doesn't work for vectors of different lengths:
L3 <- list(AB=c(6,9,2,6), BC=1:8, CD=c(-3,2) )
sapply(L3, I)

## $AB
## [1] 6 9 2 6
##
## $BC
## [1] 1 2 3 4 5 6 7 8
##
## $CD
## [1] -3  2
```

S-apply IV

```r
# the solution: berryFunctions::l2df
# list to data.frame conversion
if(!require(berryFunctions)) install.packages("berryFunctions")
library(berryFunctions)

l2df(L3)

##    V1 V2 V3 V4 V5 V6 V7 V8
## AB  6  9  2  6 NA NA NA NA
## BC  1  2  3  4  5  6  7  8
## CD -3  2 NA NA NA NA NA NA

# vectors are padded with NAs, names are kept
```

S-apply V

```
L3

## $AB
## [1] 6 9 2 6
##
## $BC
## [1] 1 2 3 4 5 6 7 8
##
## $CD
## [1] -3  2

# nested subsetting: get the third element of each vector:
sapply(L3, "[", 3)

## AB BC CD
##  2  3 NA
```

apply functions to vectors & further arguments

```
sample(x=1:100, size=5)

## [1] 17 30 33 72 56
```

apply functions to vectors & further arguments

```
sample(x=1:100, size=5)

## [1] 17 30 33 72 56
```

```
lapply(c(6,2,13), sample, x=1:100)

## [[1]]
## [1] 37 30 67 65 62 52
##
## [[2]]
## [1] 23 85
##
## [[3]]
##  [1] 91  3 21 16 12 44 74 83 53  8 42 38  5
```

Practice lists

Exercise 12: lists

1. Create a matrix with some numbers you like, with >= 2 rows and columns.
2. Create a list (with meaningful element names!) that contains the previous matrix, your name, the current `Sys.time()`, the number of this exercise, and the haircolors of at least two of your neighbours.
3. What do you get with `mean(mylist[1])`? Why?
4. Using the appropriate function from the apply family, get the `length` for each element. What is surprising about he result?
5. BONUS 1: Can a list containing another list be created?
6. BONUS 2: Change the `haircolors` element class into a category with `as.factor`. Inspect the new list with `str`.
7. tough BONUS 3: instead of using `DF[3,4]`, you can also explicitely call the subset function with `"["(DF, i=3, j=4)`. Create `datalist <- list(iris=head(iris), indo=head(Indometh), beaver=head(beaver1))`.
   How can you get the second column of each `datalist` element?
   How the third row? (Reminder: for a single DF, you need `DF[3, ]` with an empty entry for j.)

list exercise solutions

```
mat <- matrix(c(42,007,123,7777,-17,88), ncol=2)
mylist <- list(matrix=mat, creator="Berry",
     created=Sys.time(), exerciseNumber=11,
     haircolor=c("brunette", "red", "blonde", "blonde")  )
# 3 see below
sapply(mylist, length) # matrix length is 6 (=ncol*nrow)
# B1: Yes, you'll create a so-called recursive list.
mylist$haircolor <- as.factor(mylist$haircolor)
str(mylist)
sapply(datalist, "[", , 2) # or sapply(datalist, "[", 2)
sapply(datalist, "[", 3, )
```

3: `mean(mylist[1])` returns NA (along with a warning) because `mean` needs a numerical input. `mylist[1]` returns a list with a single element. `mean(mylist[[1]])` is needed instead.

apply custom functions to lists

Function to get the sum of elements 3,4,5 in a vector:

```
sum35 <- function(x) sum(x[3:5])
sum35(1:9)

## [1] 12
```

apply custom functions to lists

Function to get the sum of elements 3,4,5 in a vector:

```
sum35 <- function(x) sum(x[3:5])
sum35(1:9)

## [1] 12
```

```
L <- list(1:7, c(42,007,pi,2,4), rnorm(15), 99:96)
sapply(L, sum35)

## [1] 12.000000  9.141593  0.944905        NA
```

apply custom functions to lists

Function to get the sum of elements 3,4,5 in a vector:

```
sum35 <- function(x) sum(x[3:5])
sum35(1:9)

## [1] 12
```

```
L <- list(1:7, c(42,007,pi,2,4), rnorm(15), 99:96)
sapply(L, sum35)

## [1] 12.000000  9.141593  0.944905        NA
```

```
sapply(L, function(x) sum(x[3:5])  )
```

called "anonymous" function (not defined with a name)

list and df exercise

Exercise 13: lists and dataframes

1. Create a list with the following built-in datasets (i.e. available directly in R): `infert, Indometh, euro, esoph, VADeaths, sleep`. Use custom element names and a useful object name. Check it with `str`.
2. Now change the esoph element to keep only the numerical columns.
3. For each of the following tasks, compare `lapply` and `sapply`. If simplification doesn't work, analyze why and see if you can exclude elements of the list to make it work. For each element, get the `class, dim, ncol, NCOL, median, range, summary`.
4. Instead of using `ToothGrowth[5,2]`, you can also explicitely call the subset function with `"["(ToothGrowth, 5, 2)`.
   Reminder: for a row, you need `ToothGrowth[5, ]` with an empty entry for j.
   Get the third column of `ToothGrowth` in this explicit way.
5. Get the second column of each data.frame element of the previous list, again comparing `lapply` and `sapply`. Then get the third row of each element.

list and df exercise solution

```
manyDatasets <- list(infertility=infert, indo=Indometh,
    euro=euro, cancer=esoph, deathrates=VADeaths, sleep=sleep)
manyDatasets$cancer <- manyDatasets$cancer[,4:5]
str(manyDatasets)
lapply(manyDatasets, class)
sapply(manyDatasets, class)
sapply(manyDatasets[-2], class)
sapply(manyDatasets[-3], dim)
sapply(manyDatasets, ncol) # NULL for vector, cannot be simplified
sapply(manyDatasets, NCOL)
sapply(manyDatasets[c(3,5)], median) # other are DFs = non-numerics
sapply(manyDatasets[3:5], range) # range can handle all-numeric DFs

"["(ToothGrowth,,3)

lapply(manyDatasets[-3], "[", , j=2)
lapply(manyDatasets[-3], "[", 3, )
```

Outline

R course info
1. One-Session-Intro
2. Objects
   1. Rstudio + getting help
   2. R syntax
   3. Assignments
   4. Vectors, indexing
   5. Data.frames
   6. Matrices, apply function
   7. Lists, lapply/sapply functions
   8. Classes and methods, data types and object types
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Determine the class of an object

Everything is an object that belongs to an existing class!!

```
class(v)

## [1] "integer"

class(df)

## [1] "data.frame"

class(m)

## [1] "matrix"
```

Usage of class in functions

```
x <- 10
class(x) <- c("a", "b")
# error:
if( class(x) == "b"   ) message("class of x contains b")

## Warning in if (class(x) == "b") message("class of x
contains b"):  the condition has length > 1 and only the
first element will be used

# use this instead
if( inherits(x, "b")  ) message("good coding!")

## good coding!
```

Examining objects

Every object class has attributes and methods:

```
str(v)

##  int [1:18] 1 1 1 2 2 2 3 3 3 1 ...

str(df)

## 'data.frame': 4 obs. of  3 variables:
##  $ col_a     : int  11 12 13 14
##  $ ColName2  : Factor w/ 4 levels "a","b","c","d": 1 2 3 4
##  $ LogicalCol: logi  FALSE FALSE TRUE TRUE

str(m)

##  int [1:2, 1:3] 11 12 13 14 15 16
```

Overview of some useful functions

```
length(v)

## [1] 18

names(df)

## [1] "col_a"       "ColName2"   "LogicalCol"

dim(m)

## [1] 2 3

which(v > 4)

## integer(0)
```

Overview: data types

In order of coercion

Overview: data types

In order of coercion (if mixed, TRUE is converted to 1, 3.14 to "3.14" etc)

Overview: data types

In order of coercion (if mixed, TRUE is converted to 1, 3.14 to "3.14" etc)

Description	example	`typeof`	`class`

Overview: data types

In order of coercion (if mixed, TRUE is converted to 1, 3.14 to "3.14" etc)

Description	example	`typeof`	`class`
empty set	NULL	NULL	NULL

Overview: data types

In order of coercion (if mixed, TRUE is converted to 1, 3.14 to "3.14" etc)

Description	example	`typeof`	`class`
empty set	NULL	NULL	NULL
not available	NA	logical	logical

Overview: data types

In order of coercion (if mixed, TRUE is converted to 1, 3.14 to "3.14" etc)

Description	example	`typeof`	`class`
empty set	NULL	NULL	NULL
not available	NA	logical	logical
logical	c(T, F, FALSE, TRUE)	logical	logical

Overview: data types

In order of coercion (if mixed, TRUE is converted to 1, 3.14 to "3.14" etc)

Description	example	`typeof`	`class`
empty set	NULL	NULL	NULL
not available	NA	logical	logical
logical	c(T, F, FALSE, TRUE)	logical	logical
category	factor("left")	integer	**factor**

Overview: data types

In order of coercion (if mixed, TRUE is converted to 1, 3.14 to "3.14" etc)

Description	example	`typeof`	`class`
empty set	NULL	NULL	NULL
not available	NA	logical	logical
logical	c(T, F, FALSE, TRUE)	logical	logical
category	factor("left")	integer	**factor**
integer number	4:6 ; 7L	integer	integer

Overview: data types

In order of coercion (if mixed, TRUE is converted to 1, 3.14 to "3.14" etc)

Description	example	`typeof`	`class`
empty set	NULL	NULL	NULL
not available	NA	logical	logical
logical	c(T, F, FALSE, TRUE)	logical	logical
category	factor("left")	integer	**factor**
integer number	4:6 ; 7L	integer	integer
decimal	8.7	double	**numeric**

Overview: data types

In order of coercion (if mixed, TRUE is converted to 1, 3.14 to "3.14" etc)

Description	example	`typeof`	`class`
empty set	NULL	NULL	NULL
not available	NA	logical	logical
logical	c(T, F, FALSE, TRUE)	logical	logical
category	factor("left")	integer	**factor**
integer number	4:6 ; 7L	integer	integer
decimal	8.7	double	**numeric**
complex	5+3i	complex	complex

Overview: data types

In order of coercion (if mixed, TRUE is converted to 1, 3.14 to "3.14" etc)

Description	example	`typeof`	`class`
empty set	NULL	NULL	NULL
not available	NA	logical	logical
logical	c(T, F, FALSE, TRUE)	logical	logical
category	factor("left")	integer	**factor**
integer number	4:6 ; 7L	integer	integer
decimal	8.7	double	**numeric**
complex	5+3i	complex	complex
character string	"homer rocks"	character	character

Overview: data types

In order of coercion (if mixed, TRUE is converted to 1, 3.14 to "3.14" etc)

Description	example	`typeof`	`class`
empty set	NULL	NULL	NULL
not available	NA	logical	logical
logical	c(T, F, FALSE, TRUE)	logical	logical
category	factor("left")	integer	**factor**
integer number	4:6 ; 7L	integer	integer
decimal	8.7	double	**numeric**
complex	5+3i	complex	complex
character string	"homer rocks"	character	character
time	Sys.time()	double	**POSIXct**

Overview: data types

In order of coercion (if mixed, TRUE is converted to 1, 3.14 to "3.14" etc)

Description	example	`typeof`	`class`
empty set	NULL	NULL	NULL
not available	NA	logical	logical
logical	c(T, F, FALSE, TRUE)	logical	logical
category	factor("left")	integer	**factor**
integer number	4:6 ; 7L	integer	integer
decimal	8.7	double	**numeric**
complex	5+3i	complex	complex
character string	"homer rocks"	character	character
time	Sys.time()	double	**POSIXct**
date	as.Date("2017-05-02")	double	**Date**

Overview: data types

In order of coercion (if mixed, TRUE is converted to 1, 3.14 to "3.14" etc)

Description	example	`typeof`	`class`
empty set	NULL	NULL	NULL
not available	NA	logical	logical
logical	c(T, F, FALSE, TRUE)	logical	logical
category	factor("left")	integer	**factor**
integer number	4:6 ; 7L	integer	integer
decimal	8.7	double	**numeric**
complex	5+3i	complex	complex
character string	"homer rocks"	character	character
time	Sys.time()	double	**POSIXct**
date	as.Date("2017-05-02")	double	**Date**
function	ncol	closure	**function**

Overview: data types

In order of coercion (if mixed, TRUE is converted to 1, 3.14 to "3.14" etc)

Description	example	`typeof`	`class`
empty set	NULL	NULL	NULL
not available	NA	logical	logical
logical	c(T, F, FALSE, TRUE)	logical	logical
category	factor("left")	integer	**factor**
integer number	4:6 ; 7L	integer	integer
decimal	8.7	double	**numeric**
complex	5+3i	complex	complex
character string	"homer rocks"	character	character
time	Sys.time()	double	**POSIXct**
date	as.Date("2017-05-02")	double	**Date**
function	ncol	closure	**function**

adv-r.had.co.nz/Data-structures.

Overview: data types

In order of coercion (if mixed, TRUE is converted to 1, 3.14 to "3.14" etc)

Description	example	`typeof`	`class`
empty set	NULL	NULL	NULL
not available	NA	logical	logical
logical	c(T, F, FALSE, TRUE)	logical	logical
category	factor("left")	integer	**factor**
integer number	4:6 ; 7L	integer	integer
decimal	8.7	double	**numeric**
complex	5+3i	complex	complex
character string	"homer rocks"	character	character
time	Sys.time()	double	**POSIXct**
date	as.Date("2017-05-02")	double	**Date**
function	ncol	closure	**function**

adv-r.had.co.nz/Data-structures. `as.character(3.14)` converts a data type;

Overview: data types

In order of coercion (if mixed, TRUE is converted to 1, 3.14 to "3.14" etc)

Description	example	`typeof`	`class`
empty set	NULL	NULL	NULL
not available	NA	logical	logical
logical	c(T, F, FALSE, TRUE)	logical	logical
category	factor("left")	integer	**factor**
integer number	4:6 ; 7L	integer	integer
decimal	8.7	double	**numeric**
complex	5+3i	complex	complex
character string	"homer rocks"	character	character
time	Sys.time()	double	**POSIXct**
date	as.Date("2017-05-02")	double	**Date**
function	ncol	closure	**function**

adv-r.had.co.nz/Data-structures. `as.character(3.14)` converts a data type; `is.integer(4:6)` checks.

Overview: data types

In order of coercion (if mixed, TRUE is converted to 1, 3.14 to "3.14" etc)

Description	example	`typeof`	`class`
empty set	NULL	NULL	NULL
not available	NA	logical	logical
logical	c(T, F, FALSE, TRUE)	logical	logical
category	factor("left")	integer	**factor**
integer number	4:6 ; 7L	integer	integer
decimal	8.7	double	**numeric**
complex	5+3i	complex	complex
character string	"homer rocks"	character	character
time	Sys.time()	double	**POSIXct**
date	as.Date("2017-05-02")	double	**Date**
function	ncol	closure	**function**

adv-r.had.co.nz/Data-structures. `as.character(3.14)` converts a data type; `is.integer(4:6)` checks. `str` shows an abbreviation of `class`.

Overview: data types

In order of coercion (if mixed, TRUE is converted to 1, 3.14 to "3.14" etc)

Description	example	`typeof`	`class`
empty set	NULL	NULL	NULL
not available	NA	logical	logical
logical	c(T, F, FALSE, TRUE)	logical	logical
category	factor("left")	integer	**factor**
integer number	4:6 ; 7L	integer	integer
decimal	8.7	double	**numeric**
complex	5+3i	complex	complex
character string	"homer rocks"	character	character
time	Sys.time()	double	**POSIXct**
date	as.Date("2017-05-02")	double	**Date**
function	ncol	closure	**function**

adv-r.had.co.nz/Data-structures. `as.character(3.14)` converts a data type; `is.integer(4:6)` checks. `str` shows an abbreviation of `class`. `mode` (for users) is like `typeof` (R internal), but combines integer and double to numeric (& closure, special and builtin to function).

Overview: data types

In order of coercion (if mixed, TRUE is converted to 1, 3.14 to "3.14" etc)

Description	example	`typeof`	`class`
empty set	NULL	NULL	NULL
not available	NA	logical	logical
logical	c(T, F, FALSE, TRUE)	logical	logical
category	factor("left")	integer	**factor**
integer number	4:6 ; 7L	integer	integer
decimal	8.7	double	**numeric**
complex	5+3i	complex	complex
character string	"homer rocks"	character	character
time	Sys.time()	double	**POSIXct**
date	as.Date("2017-05-02")	double	**Date**
function	ncol	closure	**function**

adv-r.had.co.nz/Data-structures. `as.character(3.14)` converts a data type; `is.integer(4:6)` checks. `str` shows an abbreviation of `class`. `mode` (for users) is like `typeof` (R internal), but combines integer and double to numeric (& closure, special and builtin to function). Other rare typeofs: raw, environment, promise, ...

Overview: data types

In order of coercion (if mixed, TRUE is converted to 1, 3.14 to "3.14" etc)

Description	example	`typeof`	`class`
empty set	NULL	NULL	NULL
not available	NA	logical	logical
logical	c(T, F, FALSE, TRUE)	logical	logical
category	factor("left")	integer	**factor**
integer number	4:6 ; 7L	integer	integer
decimal	8.7	double	**numeric**
complex	5+3i	complex	complex
character string	"homer rocks"	character	character
time	Sys.time()	double	**POSIXct**
date	as.Date("2017-05-02")	double	**Date**
function	ncol	closure	**function**

adv-r.had.co.nz/Data-structures. `as.character(3.14)` converts a data type; `is.integer(4:6)` checks. `str` shows an abbreviation of `class`. `mode` (for users) is like `typeof` (R internal), but combines integer and double to numeric (& closure, special and builtin to function). Other rare typeofs: raw, environment, promise, ... When mixing date/time with others, the order of appearance determines the output class.

Overview: Object types

Object	example	`typeof`	`class`

Overview: Object types

Object	example	`typeof`	`class`
vector	*see data types*	...	...

Overview: Object types

Object	example	`typeof`	`class`
vector	*see data types*	...	...
matrix	matrix(9:15, ncol=2)	...	matrix

Overview: Object types

Object	example	`typeof`	`class`
vector	*see data types*	...	...
matrix	matrix(9:15, ncol=2)	...	matrix
array	array(letters[1:24], dim=c(2,6,4))	...	array

Overview: Object types

Object	example	`typeof`	`class`
vector	*see data types*	...	...
matrix	matrix(9:15, ncol=2)	...	matrix
array	array(letters[1:24], dim=c(2,6,4))	...	array
data.frame	data.frame(C1=4:5, C2=c("a","b"))	list	data.frame

Overview: Object types

Object	example	`typeof`	`class`
vector	*see data types*	...	...
matrix	matrix(9:15, ncol=2)	...	matrix
array	array(letters[1:24], dim=c(2,6,4))	...	array
data.frame	data.frame(C1=4:5, C2=c("a","b"))	list	data.frame
list	list(el1=7:15, el2="big")	list	list

Overview: Object types

Object	example	`typeof`	`class`
vector	*see data types*	...	...
matrix	matrix(9:15, ncol=2)	...	matrix
array	array(letters[1:24], dim=c(2,6,4))	...	array
data.frame	data.frame(C1=4:5, C2=c("a","b"))	list	data.frame
list	list(el1=7:15, el2="big")	list	list
function	function(x) 12+0.5*x	closure	function

Overview: Object types

Object	example	`typeof`	`class`
vector	*see data types*	...	...
matrix	matrix(9:15, ncol=2)	...	matrix
array	array(letters[1:24], dim=c(2,6,4))	...	array
data.frame	data.frame(C1=4:5, C2=c("a","b"))	list	data.frame
list	list(el1=7:15, el2="big")	list	list
function	function(x) 12+0.5*x	closure	function
...	lm(b ∼ a)	list	lm

Overview: Object types

Object	example	`typeof`	`class`
vector	*see data types*	...	...
matrix	matrix(9:15, ncol=2)	...	matrix
array	array(letters[1:24], dim=c(2,6,4))	...	array
data.frame	data.frame(C1=4:5, C2=c("a","b"))	list	data.frame
list	list(el1=7:15, el2="big")	list	list
function	function(x) 12+0.5*x	closure	function
...	lm(b ∼ a)	list	lm

A `matrix` consists of only one data type.

Overview: Object types

Object	example	`typeof`	`class`
vector	*see data types*	...	...
matrix	matrix(9:15, ncol=2)	...	matrix
array	array(letters[1:24], dim=c(2,6,4))	...	array
data.frame	data.frame(C1=4:5, C2=c("a","b"))	list	data.frame
list	list(el1=7:15, el2="big")	list	list
function	function(x) 12+0.5*x	closure	function
...	lm(b ∼ a)	list	lm

A `matrix` consists of only one data type. If you accidentally change one element to a character, all are converted

Overview: Object types

Object	example	`typeof`	`class`
vector	*see data types*	...	...
matrix	matrix(9:15, ncol=2)	...	matrix
array	array(letters[1:24], dim=c(2,6,4))	...	array
data.frame	data.frame(C1=4:5, C2=c("a","b"))	list	data.frame
list	list(el1=7:15, el2="big")	list	list
function	function(x) 12+0.5*x	closure	function
...	lm(b ∼ a)	list	lm

A `matrix` consists of only one data type. If you accidentally change one element to a character, all are converted and calculations are not possible any more (See coercion order in previous slide).

Overview: Object types

Object	example	`typeof`	`class`
vector	*see data types*	...	...
matrix	matrix(9:15, ncol=2)	...	matrix
array	array(letters[1:24], dim=c(2,6,4))	...	array
data.frame	data.frame(C1=4:5, C2=c("a","b"))	list	data.frame
list	list(el1=7:15, el2="big")	list	list
function	function(x) 12+0.5*x	closure	function
...	lm(b ∼ a)	list	lm

A `matrix` consists of only one data type. If you accidentally change one element to a character, all are converted and calculations are not possible any more (See coercion order in previous slide).
`data.frame`s can have multiple data types,

Overview: Object types

Object	example	`typeof`	`class`
vector	*see data types*	...	...
matrix	matrix(9:15, ncol=2)	...	matrix
array	array(letters[1:24], dim=c(2,6,4))	...	array
data.frame	data.frame(C1=4:5, C2=c("a","b"))	list	data.frame
list	list(el1=7:15, el2="big")	list	list
function	function(x) 12+0.5*x	closure	function
...	lm(b ∼ a)	list	lm

A `matrix` consists of only one data type. If you accidentally change one element to a character, all are converted and calculations are not possible any more (See coercion order in previous slide).
`data.frame`s can have multiple data types, but a column in itself also has only one type.

Overview: Object types

Object	example	`typeof`	`class`
vector	*see data types*	...	...
matrix	matrix(9:15, ncol=2)	...	matrix
array	array(letters[1:24], dim=c(2,6,4))	...	array
data.frame	data.frame(C1=4:5, C2=c("a","b"))	list	data.frame
list	list(el1=7:15, el2="big")	list	list
function	function(x) 12+0.5*x	closure	function
...	lm(b ∼ a)	list	lm

A `matrix` consists of only one data type. If you accidentally change one element to a character, all are converted and calculations are not possible any more (See coercion order in previous slide).
`data.frame`s can have multiple data types, but a column in itself also has only one type.
`list`s can combine anything, even other lists.

Overview: Object types

Object	example	`typeof`	`class`
vector	*see data types*	...	...
matrix	matrix(9:15, ncol=2)	...	matrix
array	array(letters[1:24], dim=c(2,6,4))	...	array
data.frame	data.frame(C1=4:5, C2=c("a","b"))	list	data.frame
list	list(el1=7:15, el2="big")	list	list
function	function(x) 12+0.5*x	closure	function
...	lm(b ∼ a)	list	lm

A `matrix` consists of only one data type. If you accidentally change one element to a character, all are converted and calculations are not possible any more (See coercion order in previous slide).
`data.frame`s can have multiple data types, but a column in itself also has only one type.
`list`s can combine anything, even other lists.
`is.atomic(Object)` returns TRUE (vector, matrix, array) or FALSE

Overview: Object types

Object	example	`typeof`	`class`
vector	*see data types*	...	...
matrix	matrix(9:15, ncol=2)	...	matrix
array	array(letters[1:24], dim=c(2,6,4))	...	array
data.frame	data.frame(C1=4:5, C2=c("a","b"))	list	data.frame
list	list(el1=7:15, el2="big")	list	list
function	function(x) 12+0.5*x	closure	function
...	lm(b ∼ a)	list	lm

A `matrix` consists of only one data type. If you accidentally change one element to a character, all are converted and calculations are not possible any more (See coercion order in previous slide).
`data.frame`s can have multiple data types, but a column in itself also has only one type.
`list`s can combine anything, even other lists.
`is.atomic(Object)` returns TRUE (vector, matrix, array) or FALSE
`as.matrix(Object)` converts the class of an object by force.

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
   1. Reading and writing data
   2. Factors, tapply function
   3. Merging data.frames
   4. Management of NAs, apply function
   5. Reproducible coding
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Data I/O functions (Input / Output)

R **reads and writes** data formats like ASCII files, GEOTIFF, GRIB, NETCDF, HDF ...

Data I/O functions (Input / Output)

R **reads and writes** data formats like ASCII files, GEOTIFF, GRIB, NETCDF, HDF ...

```
read.csv("relative/path/to/input/file.csv")
# Combined with a variable assignment:
mydata <- read.table("C:/path/to/input/file.txt")
write.table(x=mynewdata, file="output.txt")
```

Windows users: change all \ to \\ or /

Data I/O functions (Input / Output)

R **reads and writes** data formats like ASCII files, GEOTIFF, GRIB, NETCDF, HDF ...

```
read.csv("relative/path/to/input/file.csv")
# Combined with a variable assignment:
mydata <- read.table("C:/path/to/input/file.txt")
write.table(x=mynewdata, file="output.txt")
```

Windows users: change all \ to \\ or /

Relative file names (minimal changing of working directory):

```
setwd("C:/Users/berry/Desktop") # set working directory
dir() # show files in current folder
mydata <- read.table("filename.txt", header=TRUE, sep="\t")
# use a better object name!
```

Folder management

```
setwd("..") # sets working directory path up one level
getwd() # shows the current wd path
dir(recursive=TRUE) # also list items in subfolders
dir("../other_subfolder") # list files in different folder
file.create() # some file operations
file.rename()
file.remove() # see also unlink()
file.copy()
dir.create() # some folder (directory) operations
file.exists()
dir.exists()
```

Some of the arguments for `read.table`

header = TRUE	read first line as column names

Some of the arguments for `read.table`

header = TRUE	read first line as column names
dec = ","	comma as decimal mark

Some of the arguments for `read.table`

header = TRUE	read first line as column names
dec = ","	comma as decimal mark
sep = "_"	underscore as column separator ("\t" for tabstop)

Some of the arguments for `read.table`

header = TRUE	read first line as column names
dec = ","	comma as decimal mark
sep = "_"	underscore as column separator ("\t" for tabstop)
fill = T	fill incomplete rows with NAs at the end

Some of the arguments for `read.table`

header = TRUE	read first line as column names
dec = ","	comma as decimal mark
sep = "_"	underscore as column separator ("\t" for tabstop)
fill = T	fill incomplete rows with NAs at the end
skip = 12	ignore the first 12 lines (eg with meta data)

Some of the arguments for `read.table`

header = TRUE	read first line as column names
dec = ","	comma as decimal mark
sep = "_"	underscore as column separator ("\t" for tabstop)
fill = T	fill incomplete rows with NAs at the end
skip = 12	ignore the first 12 lines (eg with meta data)
comment.char = "%"	omit (rest of) lines that start with % (like R's #)

Some of the arguments for `read.table`

header = TRUE	read first line as column names
dec = ","	comma as decimal mark
sep = "_"	underscore as column separator ("\t" for tabstop)
fill = T	fill incomplete rows with NAs at the end
skip = 12	ignore the first 12 lines (eg with meta data)
comment.char = "%"	omit (rest of) lines that start with % (like R's #)
na.strings = c(-999, "NN")	identify NA entries (missing values)

Some of the arguments for `read.table`

header = TRUE	read first line as column names
dec = ","	comma as decimal mark
sep = "_"	underscore as column separator ("\t" for tabstop)
fill = T	fill incomplete rows with NAs at the end
skip = 12	ignore the first 12 lines (eg with meta data)
comment.char = "%"	omit (rest of) lines that start with % (like R's #)
na.strings = c(-999, "NN")	identify NA entries (missing values)
stringsAsFactors=FALSE	do not convert characters to factors

Some of the arguments for `read.table`

header = TRUE	read first line as column names
dec = ","	comma as decimal mark
sep = "_"	underscore as column separator ("\t" for tabstop)
fill = T	fill incomplete rows with NAs at the end
skip = 12	ignore the first 12 lines (eg with meta data)
comment.char = "%"	omit (rest of) lines that start with % (like R's #)
na.strings = c(-999, "NN")	identify NA entries (missing values)
stringsAsFactors=FALSE	do not convert characters to factors
as.is=TRUE	the same, but less typing, and potentially columnwise

Some of the arguments for `read.table`

header = TRUE	read first line as column names
dec = ","	comma as decimal mark
sep = "_"	underscore as column separator ("\t" for tabstop)
fill = T	fill incomplete rows with NAs at the end
skip = 12	ignore the first 12 lines (eg with meta data)
comment.char = "%"	omit (rest of) lines that start with % (like R's #)
na.strings = c(-999, "NN")	identify NA entries (missing values)
stringsAsFactors=FALSE	do not convert characters to factors
as.is=TRUE	the same, but less typing, and potentially columnwise

Alternatives to read.table:

Some of the arguments for `read.table`

header = TRUE	read first line as column names
dec = ","	comma as decimal mark
sep = "_"	underscore as column separator ("\t" for tabstop)
fill = T	fill incomplete rows with NAs at the end
skip = 12	ignore the first 12 lines (eg with meta data)
comment.char = "%"	omit (rest of) lines that start with % (like R's #)
na.strings = c(-999, "NN")	identify NA entries (missing values)
stringsAsFactors=FALSE	do not convert characters to factors
as.is=TRUE	the same, but less typing, and potentially columnwise

Alternatives to read.table:

scan At the core of read.table - for complicated things

Some of the arguments for `read.table`

header = TRUE	read first line as column names
dec = ","	comma as decimal mark
sep = "_"	underscore as column separator ("\t" for tabstop)
fill = T	fill incomplete rows with NAs at the end
skip = 12	ignore the first 12 lines (eg with meta data)
comment.char = "%"	omit (rest of) lines that start with % (like R's #)
na.strings = c(-999, "NN")	identify NA entries (missing values)
stringsAsFactors=FALSE	do not convert characters to factors
as.is=TRUE	the same, but less typing, and potentially columnwise

Alternatives to read.table:

- **scan** At the core of read.table - for complicated things
- **read.csv** comma separated values (different defaults than read.table)

Some of the arguments for `read.table`

header = TRUE	read first line as column names
dec = ","	comma as decimal mark
sep = "_"	underscore as column separator ("\t" for tabstop)
fill = T	fill incomplete rows with NAs at the end
skip = 12	ignore the first 12 lines (eg with meta data)
comment.char = "%"	omit (rest of) lines that start with % (like R's #)
na.strings = c(-999, "NN")	identify NA entries (missing values)
stringsAsFactors=FALSE	do not convert characters to factors
as.is=TRUE	the same, but less typing, and potentially columnwise

Alternatives to read.table:

- **scan** At the core of read.table - for complicated things
- **read.csv** comma separated values (different defaults than read.table)
- **read.fwf** fixed width formatted data

Some of the arguments for `read.table`

header = TRUE	read first line as column names
dec = ","	comma as decimal mark
sep = "_"	underscore as column separator ("\t" for tabstop)
fill = T	fill incomplete rows with NAs at the end
skip = 12	ignore the first 12 lines (eg with meta data)
comment.char = "%"	omit (rest of) lines that start with % (like R's #)
na.strings = c(-999, "NN")	identify NA entries (missing values)
stringsAsFactors=FALSE	do not convert characters to factors
as.is=TRUE	the same, but less typing, and potentially columnwise

Alternatives to read.table:

scan	At the core of read.table - for complicated things
read.csv	comma separated values (different defaults than read.table)
read.fwf	fixed width formatted data
read_excel	Excel files (install package, see github.com/hadley/readxl)

Frequent mistakes with `read.table`

Error in scan(file, what, nmax, sep, dec, quote, skip, nlines, na.strings, : line _ did not have _ elements	usually `header`, `sep` or `fill` is given incorrectly.

Frequent mistakes with `read.table`

Error in scan(file, what, nmax, sep, dec, quote, skip, nlines, na.strings, : line _ did not have _ elements	usually `header`, `sep` or `fill` is given incorrectly.
Warning message: In read.table("_.txt", _) : incomplete final line found by readTableHeader on 'C:_.txt'	Open the file and at the end, add a line break (with ENTER) = line feed = newline = carriage return.

Frequent mistakes with `read.table`

Error in scan(file, what, nmax, sep, dec, quote, skip, nlines, na.strings, : line _ did not have _ elements	usually `header`, `sep` or `fill` is given incorrectly.
Warning message: In read.table("_.txt", _) : incomplete final line found by readTableHeader on 'C:_.txt'	Open the file and at the end, add a line break (with ENTER) = line feed = newline = carriage return.
`str(DataframeIreadIn)` yields "factor" in some numeric columns	Check data file! `dec` may need to be corrected

Frequent mistakes with `read.table`

Error in scan(file, what, nmax, sep, dec, quote, skip, nlines, na.strings, : line _ did not have _ elements	usually `header`, `sep` or `fill` is given incorrectly.
Warning message: In read.table("_.txt", _) : incomplete final line found by readTableHeader on 'C:_.txt'	Open the file and at the end, add a line break (with ENTER) = line feed = newline = carriage return.
`str(DataframeIreadIn)` yields "factor" in some numeric columns	Check data file! `dec` may need to be corrected

```
as.numeric( as.character(df$col) )
as.numeric( gsub(pattern=",", replacement=".", x=df$col) )
# If you try to read data in the format "3,590.18", run
df$col <- as.numeric(gsub(",", "", df$col))
```

Frequent mistakes with `read.table`

Error in scan(file, what, nmax, sep, dec, quote, skip, nlines, na.strings, : line _ did not have _ elements	usually `header`, `sep` or `fill` is given incorrectly.
Warning message: In read.table("_.txt", _) : incomplete final line found by readTableHeader on 'C:_.txt'	Open the file and at the end, add a line break (with ENTER) = line feed = newline = carriage return.
`str(DataframeIreadIn)` yields "factor" in some numeric columns	Check data file! `dec` may need to be corrected

```
as.numeric( as.character(df$col) )
as.numeric( gsub(pattern=",", replacement=".", x=df$col) )
# If you try to read data in the format "3,590.18", run
df$col <- as.numeric(gsub(",", "", df$col))
```

Note: errors stop function execution, warnings can influence function performance and should only be ignored if you know what they really mean.

Reading real files (i.e. complicated things)

Exercise 14: Read complicated files

1. Read complicated.txt. You'll need to set several arguments.
2. BONUS: Replace the thousand-mark in the ThouSand column with an empty string (e.g. using `sub`), then convert to numeric.
3. BONUS: Work on exercise 15

The following arguments need specification:

Reading real files (i.e. complicated things)

Exercise 14: Read complicated files

1. Read complicated.txt. You'll need to set several arguments.
2. BONUS: Replace the thousand-mark in the ThouSand column with an empty string (e.g. using `sub`), then convert to numeric.
3. BONUS: Work on exercise 15

The following arguments need specification:

- file
- skip
- header
- sep
- dec

Solution reading complicated file

```
fake <- read.table("data/complicated.txt", skip=6,
                   header=TRUE, sep="\t", dec=",")
fake$ThouSand <- as.numeric(sub("`", "", fake$ThouSand))
fake

##      Group Some.Fake.Values ThouSand     Notes
## 1     Dark         82.07583    45991 Sometimes
## 2     Dark         87.82263    46508         I
## 3     Dark         71.32776    16996    cannot
## 4  Sunrise         37.49618    30692     think
## 5  Sunrise         29.21638     6245        of
## 6  Sunrise         68.23128    48544      good
## 7  Sunrise         93.35800     5148   remarks
## 8   Bright          1.05233    16323       for
## 9   Bright         78.01298     2016      fake
## 10  Cloudy         29.77000   104877      data
```

Practice reading tables and subsetting data.frames

Exercise 15: data.frames - working with tables

1. Read the file animals.txt *(rightclick **Raw**, save as)* with the correct settings for each needed argument and assign it to an object.
2. Check whether everything is read correctly with `str`.
3. Print only the first column (BONUS: in 3 different ways of subsetting).
4. `which` element of this vector is equal to (==) "Sparrow"?
5. Can the elephant fly? Find out with a logical query.
6. Find out how many legs the mammals have on average. Interpret the result.
7. BONUS 1: Use `tapply` to find the average cuteness score for each animal class.
8. BONUS 2: Read the file "days.txt" in a way that days, months, and years each have their own column. What other function besides `read.table` might be useful for that?

Solution for exercise 15.1-2: data.frames

```
# set working directory to path with files:
setwd("C:/Dokumente und Einstellungen/hydro/Desktop")
# which files are there:
dir()
# step by step, specify the arguments of read.table...
animals <- read.table("data/animals.txt", sep="\t",
                      header=TRUE, dec=",")
animals
str(animals) # ... until all numbers are num.
```

Solution for exercise 15.3-5: data.frames

```
# first column, and which is Sparrow:
animals$Name
animals[ ,"Name"] # more typing
animals[ ,1] # hard to interpret later on
animals$Name == "Sparrow"
which( animals$Name == "Sparrow" )

# Can the elephant fly? Automatic query:
animals[4,3]
animals[ animals$Name == "Elephant" , 3 ]
animals[ animals$Name == "Elephant" , "can.fly" ]
animals[ animals$Name == "Sparrow"  , "can.fly" ]
```

Solution for exercise 15.6-7: data.frames

```r
# How many legs do the mammals have on average?
animals[ animals$Class == "Mammal" , "n.legs" ]
mean( animals[ animals$Class == "Mammal" , "n.legs" ] )
# How cute are they, on average?
mean( animals[ animals$Class == "Mammal" , "cuteness" ] )

# How cute is each category of animals?
tapply(X=animals$cuteness, INDEX=animals$Class, FUN=mean)
# separates cuteness into groups, categorized by Class,
# then calculates mean for each group
```

Solution for exercise 15.8: data.frames

```r
# Read "days.txt" with columns for days, months and years
dir()
days <- read.table("days.txt", sep=".")
head(days)
str(days)

# custom names of columns
colnames(days) <- c("day", "month", "year")
head(days)

# other way:
read.fwf("data/days.txt", widths=c(2,1,2,1,4))

# usage example: return months for e.g. vegetation index:
days$month
```

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
   1. Reading and writing data
   2. Factors, tapply function
   3. Merging data.frames
   4. Management of NAs, apply function
   5. Reproducible coding
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Factors = categorical variables

```
iris$Species[63] # how factors are printed

## [1] versicolor
## Levels: setosa versicolor virginica

class(iris$Species)

## [1] "factor"

levels(iris$Species)

## [1] "setosa"     "versicolor" "virginica"

# R saves factors as numbers internally
as.numeric(iris$Species)[c(1,63,105)]

## [1] 1 2 3
```

factors, explained with animals.txt: the danger

factors, explained with animals.txt: the danger

```
ac <- read.table(file="data/animals.txt", sep="\t", header=T)$cuteness
ac # R calls  print.factor ,  see methods(print)

## [1] 1,3 6,5 9,8 6,3 5,8 4,5
## Levels: 1,3 4,5 5,8 6,3 6,5 9,8
```

factors, explained with animals.txt: the danger

```
ac <- read.table(file="data/animals.txt", sep="\t", header=T)$cuteness
ac # R calls  print.factor ,  see methods(print)

## [1] 1,3 6,5 9,8 6,3 5,8 4,5
## Levels: 1,3 4,5 5,8 6,3 6,5 9,8
```

```
class(ac); typeof(ac)

## [1] "factor"
## [1] "integer"
```

factors, explained with animals.txt: the danger

```
ac <- read.table(file="data/animals.txt", sep="\t", header=T)$cuteness
ac # R calls  print.factor ,  see methods(print)

## [1] 1,3 6,5 9,8 6,3 5,8 4,5
## Levels: 1,3 4,5 5,8 6,3 6,5 9,8
```

```
class(ac); typeof(ac)

## [1] "factor"
## [1] "integer"
```

```
as.numeric(ac) # don't ever do this!!

## [1] 1 5 6 4 3 2
```

factors, explained with animals.txt: the danger

```
ac <- read.table(file="data/animals.txt", sep="\t", header=T)$cuteness
ac # R calls  print.factor ,  see methods(print)

## [1] 1,3 6,5 9,8 6,3 5,8 4,5
## Levels: 1,3 4,5 5,8 6,3 6,5 9,8
```

```
class(ac); typeof(ac)

## [1] "factor"
## [1] "integer"
```

```
as.numeric(ac) # don't ever do this!!

## [1] 1 5 6 4 3 2
```

```
char <- as.character(ac)
as.numeric(sub(",",".", char))

## [1] 1.3 6.5 9.8 6.3 5.8 4.5
```

factors, explained with animals.txt: the solution

factors, explained with animals.txt: the solution

```
df <- read.table(file="data/animals.txt", sep="\t", header=T, as.is=TRUE)
str(df)

## 'data.frame': 6 obs. of  6 variables:
##  $ Name    : chr  "Frog" "Merl" "Doe" "Elephant" ...
##  $ Class   : chr  "Amphibian" "Bird" "Mammal" "Mammal" ...
##  $ can.fly : logi  FALSE TRUE FALSE FALSE TRUE FALSE
##  $ color   : chr  "green" "black" "brown" "gray" ...
##  $ n.legs  : int  4 2 4 4 4 2
##  $ cuteness: chr  "1,3" "6,5" "9,8" "6,3" ...
```

Now Name and cuteness are character strings instead of factors. `as.is=TRUE` is a special case of `stringsAsFactors=FALSE`: with the first you can specify the latter per column (I use it because I can type it more quickly).

factors are very useful to apply a function per group

factors are very useful to apply a function per group

```
str(PlantGrowth)

## 'data.frame': 30 obs. of  2 variables:
##  $ weight: num  4.17 5.58 5.18 6.11 4.5 4.61 5.17 4.53 5.33 5.14 ...
##  $ group : Factor w/ 3 levels "ctrl","trt1",..: 1 1 1 1 1 1 1 1 1 1 ...
```

factors are very useful to apply a function per group

```
str(PlantGrowth)
```

```
## 'data.frame': 30 obs. of  2 variables:
##  $ weight: num  4.17 5.58 5.18 6.11 4.5 4.61 5.17 4.53 5.33 5.14 ...
##  $ group : Factor w/ 3 levels "ctrl","trt1",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
tapply(X=PlantGrowth$weight, INDEX=PlantGrowth$group, FUN=mean)
```

```
##  ctrl  trt1  trt2
## 5.032 4.661 5.526
```

factors are very useful to apply a function per group

```
str(PlantGrowth)

## 'data.frame': 30 obs. of  2 variables:
##  $ weight: num  4.17 5.58 5.18 6.11 4.5 4.61 5.17 4.53 5.33 5.14 ...
##  $ group : Factor w/ 3 levels "ctrl","trt1",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
tapply(X=PlantGrowth$weight, INDEX=PlantGrowth$group, FUN=mean)

##  ctrl  trt1  trt2
## 5.032 4.661 5.526
```

This part of Hadley's book is a good intro to factors.

factors are also useful for plotting specifics

col can be a factor vector, integers correspond to `palette()`

factors are also useful for plotting specifics

col can be a factor vector, integers correspond to `palette()`

```
plot(rate~conc, data=Puromycin, col=state, pch=16, las=1)
legend("bottomright", levels(Puromycin$state), col=1:2, pch=16)
```

practice working with categories

Exercise 16: Factors, tapply

1. Plot the Puromycin dataset, but use the 3rd and 4th color of the default `palette()`
2. Using `tapply`, find out what you should feed chickens according to the measurements in `chickwts`.
3. BONUS 1: For the Puromycin dataset, use custom colors.
   Hint: create a vector with the colors, then index from it.
4. BONUS 2: In the `iris` dataset, add a row for the *Iris sintenisii*.
   Hint: add a level to the factor first.

Solution for exercise 16: factors, tapply

```
plot(rate~conc, data=Puromycin, col=as.integer(state)+2, pch=16)

mycol <- c("orange", "mediumpurple2")
plot(rate~conc, data=Puromycin, col=mycol[state], pch=16)

levels(iris$Species) <- c(levels(iris$Species), "sintensii" )
iris[151, ] <- c(6,3,5,2,"sintensii")
```

Solution for exercise 16: factors, tapply

```
plot(rate~conc, data=Puromycin, col=as.integer(state)+2, pch=16)

mycol <- c("orange", "mediumpurple2")
plot(rate~conc, data=Puromycin, col=mycol[state], pch=16)

levels(iris$Species) <- c(levels(iris$Species), "sintensii" )
iris[151, ] <- c(6,3,5,2,"sintensii")
```

```
sort(tapply(chickwts$weight, chickwts$feed, median))

## horsebean   linseed   soybean  meatmeal sunflower    casein
##     151.5     221.0     248.0     263.0     328.0     342.0

sort(tapply(chickwts$weight, chickwts$feed, min))

## horsebean   linseed  meatmeal   soybean    casein sunflower
##       108       141       153       158       216       226
```

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
   1. Reading and writing data
   2. Factors, tapply function
   3. Merging data.frames
   4. Management of NAs, apply function
   5. Reproducible coding
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Merge (join) tables

If you haven't already, now is a good time to download (and unzip!) all datasets.

Exercise 17: Merge

1. Read the files mergeApples1.txt *(rightclick **Raw**, save as)* and mergeApples2.txt into separate data.frames.
2. Use `rbind` to combine both tables line-wise and assign the output to e.g. `apples`.
3. Read mergeAge.txt into e.g. `age`.
4. Describe what `merge(age, apples, by.x="Name", by.y="Person")` does.
5. What happens if you set `all=TRUE` or `all.x=TRUE`?
6. BONUS 1: What is the difference between `cbind` and `rbind`?
7. BONUS 2: Why does `cbind` not work with age and apples?

Solution for exercise 17: Merge

```
ma1 <- read.table("data/mergeApples1.txt", header=T)
ma2 <- read.table("data/mergeApples2.txt", header=T)
apples <- rbind(ma1, ma2)
apples
age <- read.table("data/mergeAge.txt", header=T)

merge(age, apples, by.x="Name", by.y="Person")
# combine the information from both datasets
# for names that occur in both tables

merge(age, apples, by.x="Name", by.y="Person", all=T)
# Keep all entries, pad with NAs

# rbind: combine rows below each other
# cbind: bind columns beside each other
# dimensions and names must be correct
```

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
   1. Reading and writing data
   2. Factors, tapply function
   3. Merging data.frames
   4. Management of NAs, apply function
   5. Reproducible coding
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

NA-values (NA=not available)

Creating a table with **NA** values

```
# create some test data. "c" around ":" is not necessary
df <- data.frame(x= c(1:10), y=11:20 )
df[3,2] <- NA
head(df)

##   x  y
## 1 1 11
## 2 2 12
## 3 3 NA
## 4 4 14
## 5 5 15
## 6 6 16
```

NA-values

Handling of **NA** values

```
plot(df$x, df$y, pch=19, col="red")
```

NA-values

Checking for **NA** values

```
is.na(df)

##            x     y
##  [1,] FALSE FALSE
##  [2,] FALSE FALSE
##  [3,] FALSE  TRUE
##  [4,] FALSE FALSE
##  [5,] FALSE FALSE
##  [6,] FALSE FALSE
##  [7,] FALSE FALSE
##  [8,] FALSE FALSE
##  [9,] FALSE FALSE
## [10,] FALSE FALSE
```

NA-values

Checking for **NA** values

```
is.na(df)

##             x     y
##  [1,] FALSE FALSE
##  [2,] FALSE FALSE
##  [3,] FALSE  TRUE
##  [4,] FALSE FALSE
##  [5,] FALSE FALSE
##  [6,] FALSE FALSE
##  [7,] FALSE FALSE
##  [8,] FALSE FALSE
##  [9,] FALSE FALSE
## [10,] FALSE FALSE
```

```
na.omit(df)

##     x  y
## 1   1 11
## 2   2 12
## 4   4 14
## 5   5 15
## 6   6 16
## 7   7 17
## 8   8 18
## 9   9 19
## 10 10 20
```

NA-values

```
##     x  y
## 1   1 11
## 2   2 12
## 3   3 NA
## 4   4 14
## 5   5 15
## 6   6 16
## 7   7 17
## 8   8 18
## 9   9 19
## 10 10 20
```

NA-values

```
##     x  y
## 1   1 11
## 2   2 12
## 3   3 NA
## 4   4 14
## 5   5 15
## 6   6 16
## 7   7 17
## 8   8 18
## 9   9 19
## 10 10 20
```

```r
mean(df$x) ; mean(df$y)
```

```
## [1] 5.5
## [1] NA
```

NA-values

```
##     x  y
## 1   1 11
## 2   2 12
## 3   3 NA
## 4   4 14
## 5   5 15
## 6   6 16
## 7   7 17
## 8   8 18
## 9   9 19
## 10 10 20
```

```r
mean(df$x) ; mean(df$y)
```

```
## [1] 5.5
## [1] NA
```

```r
mean(df$y, na.rm=TRUE) # dangerous with sums
```

```
## [1] 15.77778
```

apply: execute a function on each row (col) of a `matrix`

apply: execute a function on each row (col) of a matrix

```
# Compute the column- and row sums of a data.frame (df)
apply(X=df, MARGIN=2, FUN=sum) # column wise

##  x  y
## 55 NA
```

apply: execute a function on each row (col) of a matrix

```r
# Compute the column- and row sums of a data.frame (df)
apply(X=df, MARGIN=2, FUN=sum) # column wise

##  x  y
## 55 NA
```

```r
apply(df, 1, sum) # line by line

##  [1] 12 14 NA 18 20 22 24 26 28 30
```

apply: execute a function on each row (col) of a matrix

```
# Compute the column- and row sums of a data.frame (df)
apply(X=df, MARGIN=2, FUN=sum) # column wise

##  x  y
## 55 NA
```

```
apply(df, 1, sum) # line by line

##  [1] 12 14 NA 18 20 22 24 26 28 30
```

```
apply(df, 2, sum, na.rm=TRUE) # with further arguments

##   x   y
##  55 142
```

apply: execute a function on each row (col) of a matrix

```
# Compute the column- and row sums of a data.frame (df)
apply(X=df, MARGIN=2, FUN=sum) # column wise

##  x  y
## 55 NA
```

```
apply(df, 1, sum) # line by line

##  [1] 12 14 NA 18 20 22 24 26 28 30
```

```
apply(df, 2, sum, na.rm=TRUE) # with further arguments

##   x   y
##  55 142
```

Exercise 18: apply

Compute the median of each column in the dataset `iris`.
Hint: leave out the non-numeric column.

apply practice solution

```
apply(iris[ ,-5], MARGIN=2, median)

## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##         5.80         3.00         4.35         1.30
```

apply practice solution

```
apply(iris[ ,-5], MARGIN=2, median)

## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##         5.80         3.00         4.35         1.30
```

```
apply(iris[ ,-5], MARGIN=2, mean)

## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##     5.843333     3.057333     3.758000     1.199333
```

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
   1. Reading and writing data
   2. Factors, tapply function
   3. Merging data.frames
   4. Management of NAs, apply function
   5. Reproducible coding
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Working with Scripts

For any real data analysis, you should work with scripts that save your code.

Working with Scripts

For any real data analysis, you should work with scripts that save your code. For that reason, you don't have to save your workspace when you close R.

Working with Scripts

For any real data analysis, you should work with scripts that save your code.
For that reason, you don't have to save your workspace when you close R.
If you save it, you don't know where it is stored by default.
In the next R session, all objects will be available again, which is convenient, but dangerous, as it reduces reproducibility.

Working with Scripts

For any real data analysis, you should work with scripts that save your code.
For that reason, you don't have to save your workspace when you close R.
If you save it, you don't know where it is stored by default.
In the next R session, all objects will be available again, which is convenient, but dangerous, as it reduces reproducibility.
Scripts can easily be exchanged with research partners, clients or employers.
They may also develop into packages...

Working with Scripts

For any real data analysis, you should work with scripts that save your code.
For that reason, you don't have to save your workspace when you close R.
If you save it, you don't know where it is stored by default.
In the next R session, all objects will be available again, which is convenient, but dangerous, as it reduces reproducibility.
Scripts can easily be exchanged with research partners, clients or employers.
They may also develop into packages...
Before sending, close R, open a clean (!) session, and check whether everything is running.

Working with Scripts

For any real data analysis, you should work with scripts that save your code.
For that reason, you don't have to save your workspace when you close R.
If you save it, you don't know where it is stored by default.
In the next R session, all objects will be available again, which is convenient, but dangerous, as it reduces reproducibility.
Scripts can easily be exchanged with research partners, clients or employers.
They may also develop into packages...
Before sending, close R, open a clean (!) session, and check whether everything is running.
Scripts should have lots of comments!

Working with Scripts

For any real data analysis, you should work with scripts that save your code.
For that reason, you don't have to save your workspace when you close R.
If you save it, you don't know where it is stored by default.
In the next R session, all objects will be available again, which is convenient, but dangerous, as it reduces reproducibility.
Scripts can easily be exchanged with research partners, clients or employers.
They may also develop into packages...
Before sending, close R, open a clean (!) session, and check whether everything is running.
Scripts should have lots of comments! `# seriously.`

Creating documents, Organizing your files

Rstudio - file - new -
- R HTML (no LaTeXrequired on system)
- R markdown (easy to learn, creates .pdf or .docx)
- R sweave (very powerful through LaTeX. This is what I use for these slides.)

Creating documents, Organizing your files

Rstudio - file - new -
- R HTML (no LaTeXrequired on system)
- R markdown (easy to learn, creates .pdf or .docx)
- R sweave (very powerful through LaTeX. This is what I use for these slides.)

I use knitr, as it is more advanced than the default sweave. For that, set:
Rstudio - tools - global options - Sweave - weave using: knitr
Save as .Rnw file. See the knitr demo files: yihui.name/knitr/demo

Creating documents, Organizing your files

Rstudio - file - new -
- R HTML (no LaTeXrequired on system)
- R markdown (easy to learn, creates .pdf or .docx)
- R sweave (very powerful through LaTeX. This is what I use for these slides.)

I use knitr, as it is more advanced than the default sweave. For that, set:
Rstudio - tools - global options - Sweave - weave using: knitr
Save as .Rnw file. See the knitr demo files: yihui.name/knitr/demo

There are tips online on how to organize data, scripts, plots etc.
stackoverflow: How to organize large R programs
stackexchange: How to efficiently manage a statistical analysis project?
Here are some tips by Hadley: http://r-pkgs.had.co.nz/style.html

Real Datasets online

- NOAA (USA) weather data: www1.ncdc.noaa.gov/pub/data/normals
- DWD Climate Data Center: CDC/observations_germany/climate

```
library(rdwd) # github.com/brry/rdwd
clim <- dataDWD(selectDWD(id=02080, res="daily", var="kl",
                          per="hist"), dir="DWDdata")
```

- data.fivethirtyeight.com
- data.unicef.org (potentially add .xls etc as file extension)
- github.com/okulbilisim/awesome-datascience#data-sets
- github.com/caesar0301/awesome-public-datasets
- ropensci.org/packages/index.html
- dataviz.tools/category/data-sources/
- StorytellingWithData dataviz challenge 2018

Exercise 19: Real Data

Download, read and (visually) analyze a dataset that interests you.

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
   1. Syntax, adjustments, labeling
   2. Lines
   3. Limits, adding to graphics
   4. Boxplots, histograms
   5. Saving plots, multiple figures
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Plot I: the dataset

```
head(beaver2, 4)

##   day time  temp activ
## 1 307  930 36.58     0
## 2 307  940 36.73     0
## 3 307  950 36.93     0
## 4 307 1000 37.15     0

str(beaver2)

## 'data.frame': 100 obs. of  4 variables:
##  $ day  : num  307 307 307 307 307 307 307 307 307 307 ...
##  $ time : num  930 940 950 1000 1010 1020 1030 1040 1050 1100 ...
##  $ temp : num  36.6 36.7 36.9 37.1 37.2 ...
##  $ activ: num  0 0 0 0 0 0 0 0 0 0 ...
```

Plot II

```r
# The basic syntax is plot(x, y, adjustments)
plot(beaver2$time, beaver2$temp)
#
```

Plot III

```
plot(beaver2$time, beaver2$temp, ylab="Body temperature  [ \U00B0 C]",
     main="Beavers get warm late in the evening",
     xlab="Observation time [0330 for 3:30AM]", col="red")
```

Plot IV

```r
plot(beaver2$time, beaver2$temp, pch=2, cex=2.8, las=1)
# pch: PointCHaracter, cex: CharacterEXpansion
# las: LabelAxisStyle (numbers upright)
```

Overview of Point CHaracters (more in bF Anhang)

plot (x, y, pch = _)

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
   1. Syntax, adjustments, labeling
   2. Lines
   3. Limits, adding to graphics
   4. Boxplots, histograms
   5. Saving plots, multiple figures
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Plot with lines I

```
k <- c(3,0,-10,8,6,3)   ; l <- c(-4,2,-6,-12,7,-3)
plot(k)  # standard (default) type: "p" for points
```

Plot with lines II

```
plot(k, type="l")
# type="l" for lines
```

Plot with lines III

```
plot(k, type="b", lwd=4)
# type="b" for both points and lines. # lwd: line width
```

Plot with lines IV

```
plot(k, type="o", lty=2)
# type="o" for both overplotted. # lty: line type
```

Overview of types (more in bF Anhang)

plot (x, y, type = _)

Overview of line types (more in bF Anhang)

plot (x, y, lty = _)

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
   1. Syntax, adjustments, labeling
   2. Lines
   3. Limits, adding to graphics
   4. Boxplots, histograms
   5. Saving plots, multiple figures
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

One data.frame instead of two vectors

Default method if x is a data.frame:

```
plot( beaver2[ ,2:3] )
```

Plot limits I

xlim with a two-value vector with lower and upper limit.

```
plot(beaver2[ ,2:3], xlim=c(-1000,3000),  ylim=c(36,40))
```

Plot limits II

xaxis decreasing (can be very misleading!):

```
plot(beaver2[ ,2:3], xlim=c(3000,-1000), ylim=c(36,40))
```

Adding datapoints

```
plot(beaver2[ ,2:3])
points(100, 37.5, pch=21, bg=5, cex=4, col="red", lwd=3)
lines(x=c(5,120,514,918), y=c(37,39,38,37.5), col="green4", type="b")
```

Now it's your turn...

Exercise 20: Plotting

1. Plot a dataset of your choice (eg from `library(help = "datasets")`).
2. Add some red dotted lines between big quadratic points, filled with blue color.
3. Label the axis and give your graph a big italic header with `cex.main`, `font.main`, see `?par`.
4. Plot with a logarithmic scale on one axis.
5. Set the axis to a certain limit. With `yaxs`, suppress the default 4% range extension.
6. Using `xaxt`, plot without an x-axis. Look in the help of `par` how this can be done. Then add a custom `axis`.
7. Add a `legend` to the plot and thick-lined `box` around the final plot.

Solution for exercise 20: Plotting I

```
# generate "random" data:
set.seed(42) ; numbers <- cumsum(rnorm(40)) + 20
x <- c(10, 15, 31, 33); y <- c(19, 22, 17, 23)
# Basic plot:
plot(numbers, type="l", xlab="Time",
     main="Fake Time Series", cex.main=3, font.main=3,
     log="y", xlim=c(0,50), xaxs="i", xaxt="n" )
# Add points and lines
points(x=x, y=y, pch=22, type="p", cex=3, col="orange",
      bg="green")
lines(x=c(10, 15, 31, 33), y=c(-6, -3, -8, -2)+25, col="red",
      lty="dashed")
# Custom axis, legend and box around plot:
axis(side=1, at=c(0, 15, 30, 45)  )
legend("topright", legend=c("ts", "more fake stuff"),
       lty=c(1,2), col=c(1,2) )
box(lwd=5)
```

Solution for exercise 20: Plotting II

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
   1. Syntax, adjustments, labeling
   2. Lines
   3. Limits, adding to graphics
   4. Boxplots, histograms
   5. Saving plots, multiple figures
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Health data, county-aggregated for >3k regions in the US.

Found through reddit: 18 Free Life Sciences, Healthcare and Medical Datasets
Community Health Status Indicators (CHSI) to Combat Obesity, Heart Disease and Cancer
CHSI_meta.csv abstracted from DATAELEMENTDESCRIPTION.csv
CHSI_data.csv which is RISKFACTORSANDACCESSTOCARE.csv
Remember you can simply update the material through `git pull`

Health data, county-aggregated for >3k regions in the US.

Found through reddit: 18 Free Life Sciences, Healthcare and Medical Datasets
Community Health Status Indicators (CHSI) to Combat Obesity, Heart Disease and Cancer
CHSI_meta.csv abstracted from DATAELEMENTDESCRIPTION.csv
CHSI_data.csv which is RISKFACTORSANDACCESSTOCARE.csv
Remember you can simply update the material through `git pull`

Exercise 21: medical boxplots

1. Read both files. You'll need `na.strings=c(-1111.1,-2222, -2222.2)` for the data.
2. Reduce the dataset to the following columns: `DF[,c(7,10,13,16,19,22,25:31)]`.
3. Create a boxplot of any column that sounds interesting.
4. Create a histogram of one of the first 5 columns. Why does it look like a normal distribution? (I told you these don't usually occur in real life.)
5. Create a boxplot of the first 5 columns to compare percentages.
6. BONUS: Create a violin plot with the `vioplot` package (or `ggplot2::geom_violin`).
7. Using the formula interface (`Y X, data=someDF`), create a boxplot showing the **number of doctors per 100k people** (Prim_Care_Phys_Rate) depending on whether it's a designated **health professional shortage area** (HPSA_Ind).
8. BONUS: Create a histogram of the number of uninsured people per county. Why does the distribution look like this? See if `berryFunctions` has a way to create a histogram on a logarithmic axis. Contemplate the concept of a lognormal distribution in comparison to an exponential distribution.

solutions medical boxplots

```
meta <- read.csv("data/CHSI_meta.csv")
health <- read.csv("data/CHSI_data.csv",
                   na.strings=c(-1111.1,-2222, -2222.2))
health <- health[,c(7,10,13,16,19,22,25:31)]
meta   <-   meta[ c(7,10,13,16,19,22,25:31), ]

hist(health$Obesity, breaks=30)
# Central limit theorem: aggregation -> normality
# We're looking at percentage of obesity per county,
# not at individuals
boxplot(health[,1:5])
vioplot::vioplot(health[,1:5])
boxplot(Prim_Care_Phys_Rate~HPSA_Ind, data=health)

hist(health$Uninsured)
berryFunctions::logHist(health$Uninsured, breaks=20)
plot(1:100, dexp(1:100, rate=0.06), col="red", type="l")
lines(1:100, dlnorm(1:100, 3, 0.6), col="blue")
```

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
   1. Syntax, adjustments, labeling
   2. Lines
   3. Limits, adding to graphics
   4. Boxplots, histograms
   5. Saving plots, multiple figures
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Saving plots to a file

In the menu of the graphics window, you can save the current plot by hand. For reproducible sizes and if you're creating many plots, use

Saving plots to a file

In the menu of the graphics window, you can save the current plot by hand. For reproducible sizes and if you're creating many plots, use

```
png("FileName.png", width=12, height=9, units="in",
           res=300, bg="transparent", pointsize=14)
  plot(1:20)
  title(main="Plot title")
  points(5,9)
dev.off() # to close the external device
```

Saving plots to a file

In the menu of the graphics window, you can save the current plot by hand. For reproducible sizes and if you're creating many plots, use

```
png("FileName.png", width=12, height=9, units="in",
           res=300, bg="transparent", pointsize=14)
  plot(1:20)
  title(main="Plot title")
  points(5,9)
dev.off() # to close the external device
```

The functions `pdf`, `jpeg`, etc. are also available.

multiple figure layout I

```
# 3 rows, 2 columns:
par(mfrow=c(3,2), mar=c(3,3,1,1), mgp=c(1.9, 0.7, 0), las=1)
plot(stackloss[ ,1:2]) # 6 plot commands
```

multiple figure layout II

```
lay <- layout(matrix(c(1,1,2,2,
                       5,5,2,2,
                       3,3,4,4,
                       3,3,4,4), ncol=4, byrow=TRUE),
              widths=c(6,6,4,4))
layout.show(lay)
```

multiple figure layout III

From the example in `?layout`:

Box

Y−Axis
0.8
0.6
0.4
0.2
0.0
1
2
3
4
5
6
box(_)
1.4
1.0
0.6
'plot'
'figure', after par(mfrow) or layout()
'inner'
'outer', if outer margins were set by par(oma)

Enough talk, action...

Exercise 22: Export (multipanel) plots

1. Read Zugspitze.txt *(rightclick **Raw**, save as)* with monthly weather records
2. With `par(mfrow=...)`, set up a figure with two panels (left and right)
3. Plot a histogram of the air temp (MO_TT) records in the left panel
4. Plot a line graph of the first 50 months in the right panel
5. Now export this graph as png with useful size and resolution
6. BONUS 1: also export as a vector graph (e.g. PDF, EPS, SVG, svglite package)
7. BONUS 2: Set smaller margins (`mar`) around each panel
8. BONUS 3: Set an outer margin (`oma`) with an outer `title` for the entire figure

Solution for exercise 22: Export (multipanel) plots I

```
wzug <- read.table("data/Zugspitze.txt", header=TRUE)

pdf("fig/zugspitze.pdf", height=4)
par(mfrow=c(1,2), mar=c(3,3,1,1), oma=c(0,0,2,0), las=1)
hist(wzug$MO_TT, col="coral", breaks=25,
     main="Histogram monthly values")
plot(wzug$MO_TT[1:50], type="l", lwd=2,
     col="darkorchid", main="First 50 months")
title(main="Temperature Zugspitze 1900:2015", outer=TRUE)
dev.off()

## RStudioGD
##         2
```

Solution for exercise 22: Export (multipanel) plots II

Graphing paths: NA lines between entry groups

```
library("berryFunctions") # >= 1.9.2 (2015-10-22) for insertRows, colPoints
tfile <- system.file("extdata/rivers.txt", package="berryFunctions")
rivers <- read.table(tfile, header=TRUE, dec=",")
colPoints(x,y,n, data=rivers, add=FALSE, lines=TRUE, lwd=3,
          legargs=c(y1=0.8,y2=1, density=FALSE) )
```

Base R vs ggplot2

This has all been graphing with the possibilities of base R.
There is also the package `ggplot2` which a lot of people love to use.
I get everything done with base R and don't see a need to learn ggplot grammar.
You can do whatever you want - it's both R, both have their advantages.
A very good place to start learning ggplot2 is:
http://minimaxir.com/2017/08/ggplot2-web/
To compare both paradigms, see:
http://flowingdata.com/2016/03/22/comparing-ggplot2-and-r-base-graphics/
Be sure to follow the links in the beginning of the article...

Practice multipanel plots + empirical distribution analysis

Exercise 23: Boxplots and histograms

1. Read box.txt into R. Remember you can simply update the material through `git pull`.
2. Set up a multipanel plot and compare the boxplots of the three columns. How do they differ?
3. Compare the histograms of the three columns. How do they differ? Use the same axis limits on each panel.
4. Arrange all 6 panels into one graph.
5. Use a horizontal boxplot. Use the same axis limits (note: for horizontal, you still need to set `ylim`, not `xlim`).
6. BONUS: Write a loop to create the boxplots and histograms (note: you might need to change from `mfrow` to `mfcol` or vice versa).
7. BONUS: Make the graphs look nice through the use of colors, histogram breaks, nice spacing between the panels (`par mar`), an informative overall graph `title` in the outer margin (`par oma`) etc.
8. Change the histogram y axis from count to density (using the `freq` argument). Now add a line to each histogram with the density function of the normal distribution parametrized by the `mean` and `sd` of each sample. `dnorm` will need a vector of values like `0:60`.

Solution multipanel distribution plotting

```
# Read data, set colors + multipanel plot:
box <- read.table("data/box.txt", header=TRUE)
colors <- c("salmon","brown","firebrick1")
par(mfrow=c(3,2), mar=c(2,0,0.4,0.4), oma=c(0,0,2,0), cex=0.7)

# for each column, create two plots:
for(i in 1:3)
  {
  # 1: Boxplot:
  boxplot(box[,i], ylim=c(2.3,60), horizontal=TRUE, col=colors[i])
  # 2: Histogram:
  hist(box[,i], xlim=c(2.3,60), breaks=-3:62, freq=FALSE, main="",
       yaxt="n", col=colors[i])
  # add density of normal distribution parametrized to data sample
  lines(0:65, dnorm(0:65, mean(box[,i]), sd(box[,i])))
}
# Add title to outer margin (par oma):
title(main="same boxplots, different distributions!", outer=TRUE)
```

same boxplots, different distributions!

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
   1. Character strings: Intro
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Character strings I

```
paste("NiceThing", 3:5) # Creates vector of Mode character

## [1] "NiceThing 3" "NiceThing 4" "NiceThing 5"

paste(2, "hot", 4, "U", sep="_")

## [1] "2_hot_4_U"

letvec <- c("some different", "let", "ters")
letvec

## [1] "some different" "let"
## [3] "ters"
```

Character strings II

```
length(letvec)

## [1] 3

nchar(letvec)

## [1] 14  3  4

paste("nice", letvec, sep="_")

## [1] "nice_some different" "nice_let"
## [3] "nice_ters"
```

Character strings III

```
paste(letvec, sep="")

## [1] "some different" "let"
## [3] "ters"

paste(letvec, collapse="")

## [1] "some differentletters"

# remove spaces: substitute " " in letvec with ""
sub(" ", "", letvec)

## [1] "somedifferent" "let"
## [3] "ters"
```

Character strings IV

```
substr("Monster Party", start=2, stop=5)

## [1] "onst"

# If you need several string segments:
substring("Monster Party", first=c(1,3,5), last=c(2,5,9) )

## [1] "Mo"    "nst"   "ter P"

strsplit("Die Aerzte - Monster Party", split=" - ")[[1]]

## [1] "Die Aerzte"    "Monster Party"
```

Backslash "\" : Escape Character

"\n" for return or line break (newline), eg in graphictitles and -texts
"\t" for Tabstop, e.g. for `write.table("path/file.txt", yourdata, sep="\t")`
"\\" for Backslash as symbol in `title, text, legend` etc.
"\"" for " as character
"\U" for Unicode, e.g.:

```
plot(1)
# average symbol without using "expression":
mtext(paste("\U00D8", 5.4, "\U00B0 C"), col="red")
```

Time to practice...

Exercise 24: character string operations

In the German chocolate industry, we love

```
schoko <- c("Ritter Sport", "Lindor", "Frigor", "Milka", "Frey",
"More Lindor")
```

1. which element **is** or **contains** "Lindor"?
   Find out with `match`, `grep`, `%in%` and `grepl`. What are the differences between those?
2. Replace "or" with "nicor" using `sub` and `gsub`.
3. What's the problem with `"Ritter sport" %in% schoko`?
4. What does `"Ritter [s,S]port" %in% schoko` do?

Solution for exercise 24.1: Character strings

```
schoko <- c("Lindor", "Ritter Sport", "Frigor", "Milka",
            "Frey", "More Lindor")
# Which element in schoko IS Lindor
match("Lindor", schoko)
match(schoko, "Lindor")
match(schoko, c("Lindor", "Frigor") )

# Which element in schoko CONTAINS Lindor
grep("Lindor", schoko)
grep("or", schoko)
grepl("Lindor", schoko) # logical value

# Is one of the elements of schoko == "Lindor" at all?
"Lindor" %in% schoko
any(  grepl("Lindor", schoko)  ) # any TRUE?
all(  grepl("Lindor", schoko)  ) # All TRUE
! grepl("Lindor", schoko)
```

Solution for exercise 24.2-41: Character strings

```r
# Replace Character strings (substitute)
sub(pattern="or", replacement="nicor", schoko)
# Replace all occurences (global substitution)
gsub(pattern="or", replacement="nicor", schoko)

# Avoid capitalization issues with
tolower(schoko)
"Ritter [s,S]port" %in% schoko
# Does not work!!
```

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
   1. Conditions
   2. For Loops
   3. lapply, Rcpp
   4. Repeat + While Loops
   5. Arrays
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Conditional execution I

If you follow along with the handout version (one page per slide), please don't view this slide yet. We'll reveal piecewise.

Conditional execution I

If you follow along with the handout version (one page per slide), please don't view this slide yet. We'll reveal piecewise.

Syntax for a single logical value:

```
if(this_is_true) {do_something} else {do_other_thing}
```

Conditional execution I

If you follow along with the handout version (one page per slide), please don't view this slide yet. We'll reveal piecewise.

Syntax for a single logical value:

```
if(this_is_true) {do_something} else {do_other_thing}
```

Syntax for a vector with several T/F values:

```
ifelse(condition, expression1, expression2)
```

Conditional execution I

If you follow along with the handout version (one page per slide), please don't view this slide yet. We'll reveal piecewise.

Syntax for a single logical value:

```
if(this_is_true) {do_something} else {do_other_thing}
```

Syntax for a vector with several T/F values:

```
ifelse(condition, expression1, expression2)
```

If condition == TRUE, then expression1 is evaluated, if condition == FALSE, then expression2 is evaluated.

Conditional execution I

If you follow along with the handout version (one page per slide), please don't view this slide yet. We'll reveal piecewise.

Syntax for a single logical value:

```
if(this_is_true) {do_something} else {do_other_thing}
```

Syntax for a vector with several T/F values:

```
ifelse(condition, expression1, expression2)
```

If condition == TRUE, then expression1 is evaluated, if condition == FALSE, then expression2 is evaluated.

```
7-3 > 2
class(7-3 > 2 )
if(7-3 > 2) 18
if(7-3 > 5) 18
if(7-3 > 5) 18 else 17
```

Conditional execution I

If you follow along with the handout version (one page per slide), please don't view this slide yet. We'll reveal piecewise.

Syntax for a single logical value:

```
if(this_is_true) {do_something} else {do_other_thing}
```

Syntax for a vector with several T/F values:

```
ifelse(condition, expression1, expression2)
```

If condition == TRUE, then expression1 is evaluated, if condition == FALSE, then expression2 is evaluated.

```
7-3 > 2
class(7-3 > 2 )
if(7-3 > 2) 18
if(7-3 > 5) 18
if(7-3 > 5) 18 else 17
```

TRUE

Conditional execution I

If you follow along with the handout version (one page per slide), please don't view this slide yet. We'll reveal piecewise.

Syntax for a single logical value:

```
if(this_is_true) {do_something} else {do_other_thing}
```

Syntax for a vector with several T/F values:

```
ifelse(condition, expression1, expression2)
```

If condition == TRUE, then expression1 is evaluated, if condition == FALSE, then expression2 is evaluated.

```
7-3 > 2
class(7-3 > 2 )
if(7-3 > 2) 18
if(7-3 > 5) 18
if(7-3 > 5) 18 else 17
```

TRUE

logical = truth value, boolean

Conditional execution I

If you follow along with the handout version (one page per slide), please don't view this slide yet. We'll reveal piecewise.

Syntax for a single logical value:

```
if(this_is_true) {do_something} else {do_other_thing}
```

Syntax for a vector with several T/F values:

```
ifelse(condition, expression1, expression2)
```

If condition == TRUE, then expression1 is evaluated, if condition == FALSE, then expression2 is evaluated.

`7-3 > 2`	TRUE
`class(7-3 > 2 )`	logical = truth value, boolean
`if(7-3 > 2) 18`	Condition is TRUE, so 18 is returned
`if(7-3 > 5) 18`	
`if(7-3 > 5) 18 else 17`	

Conditional execution I

If you follow along with the handout version (one page per slide), please don't view this slide yet. We'll reveal piecewise.

Syntax for a single logical value:

```
if(this_is_true) {do_something} else {do_other_thing}
```

Syntax for a vector with several T/F values:

```
ifelse(condition, expression1, expression2)
```

If condition == TRUE, then expression1 is evaluated, if condition == FALSE, then expression2 is evaluated.

`7-3 > 2`	TRUE
`class(7-3 > 2 )`	logical = truth value, boolean
`if(7-3 > 2) 18`	Condition is TRUE, so 18 is returned
`if(7-3 > 5) 18`	Condition is FALSE, so nothing happens*
`if(7-3 > 5) 18 else 17`	

Conditional execution I

If you follow along with the handout version (one page per slide), please don't view this slide yet. We'll reveal piecewise.

Syntax for a single logical value:

```
if(this_is_true) {do_something} else {do_other_thing}
```

Syntax for a vector with several T/F values:

```
ifelse(condition, expression1, expression2)
```

If condition == TRUE, then expression1 is evaluated, if condition == FALSE, then expression2 is evaluated.

`7-3 > 2`	TRUE
`class(7-3 > 2 )`	logical = truth value, boolean
`if(7-3 > 2) 18`	Condition is TRUE, so 18 is returned
`if(7-3 > 5) 18`	Condition is FALSE, so nothing happens*
`if(7-3 > 5) 18 else 17`	Condition FALSE, so 17 is returned.

Conditional execution I

If you follow along with the handout version (one page per slide), please don't view this slide yet. We'll reveal piecewise.

Syntax for a single logical value:

```
if(this_is_true) {do_something} else {do_other_thing}
```

Syntax for a vector with several T/F values:

```
ifelse(condition, expression1, expression2)
```

If condition == TRUE, then expression1 is evaluated, if condition == FALSE, then expression2 is evaluated.

`7-3 > 2`	TRUE
`class(7-3 > 2 )`	logical = truth value, boolean
`if(7-3 > 2) 18`	Condition is TRUE, so 18 is returned
`if(7-3 > 5) 18`	Condition is FALSE, so nothing happens*
`if(7-3 > 5) 18 else 17`	Condition FALSE, so 17 is returned.

**: technically, `NULL` is returned invisibly*

Conditional execution II

```
# Several commands must be held together with curly braces:
if(TRUE)
  {
  plot(1:10, main="TRUE in code")
  box("figure", col="blue", lwd=3)
  } else
# do something else: plot random numbers
    {
    plot(rnorm(500), main="FALSE in code")
    }
# these last brackets can (but should not) be left away
# indenting code makes it readable for humans

par(mfrow=c(1,2), cex=1, las=1)
```

Conditional execution III

Vectorising: if(c) e1 else e2 vs ifelse(c, e1, e2)

```
v <- c(13, 14, 15, 16, 17)
v>14

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

Vectorising: if(c) e1 else e2 vs ifelse(c, e1, e2)

```
v <- c(13, 14, 15, 16, 17)
v>14

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

```
ifelse(v>14, v+10, NA) # can handle vector of input

## [1] NA NA 25 26 27
```

Vectorising: if(c) e1 else e2 vs ifelse(c, e1, e2)

```
v <- c(13, 14, 15, 16, 17)
v>14

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

```
ifelse(v>14, v+10, NA) # can handle vector of input

## [1] NA NA 25 26 27
```

```
if(v>14) v+10 else NA

## Warning in if (v > 14) v + 10 else NA: the condition has
length > 1 and only the first element will be used

## [1] NA
```

Notes on logical values

- as you might have seen in `read.table(header=T)`, logical values (TRUE, FALSE) can be abbreviated.

Notes on logical values

- as you might have seen in `read.table(header=T)`, logical values (TRUE, FALSE) can be abbreviated.
- If you want to play a mean prank on someone, write `T <- FALSE; F <- TRUE` in their Rprofile (see `?Startup`).

Notes on logical values

- as you might have seen in `read.table(header=T)`, logical values (TRUE, FALSE) can be abbreviated.
- If you want to play a mean prank on someone, write `T <- FALSE; F <- TRUE` in their Rprofile (see `?Startup`).
- Logical (boolean) values F and T internally are often converted to integers 0 and 1, thus `sum(c(T,F,F,T,T,T,F,F,T))` is the number of TRUEs in a vector, `mean` yields the proportion of TRUEs.

Notes on logical values

- as you might have seen in `read.table(header=T)`, logical values (TRUE, FALSE) can be abbreviated.
- If you want to play a mean prank on someone, write `T <- FALSE; F <- TRUE` in their Rprofile (see `?Startup`).
- Logical (boolean) values F and T internally are often converted to integers 0 and 1, thus `sum(c(T,F,F,T,T,T,F,F,T))` is the number of TRUEs in a vector, `mean` yields the proportion of TRUEs.
- `which(logicalVec)` gives the indices (positions) of TRUE values.

Notes on logical values

- as you might have seen in `read.table(header=T)`, logical values (TRUE, FALSE) can be abbreviated.
- If you want to play a mean prank on someone, write `T <- FALSE; F <- TRUE` in their Rprofile (see `?Startup`).
- Logical (boolean) values F and T internally are often converted to integers 0 and 1, thus `sum(c(T,F,F,T,T,T,F,F,T))` is the number of TRUEs in a vector, `mean` yields the proportion of TRUEs.
- `which(logicalVec)` gives the indices (positions) of TRUE values.
- `Vec[logicalVec]` returns only the values of vec corresponding to TRUE in logicalVec (No need to wrap it into `which`).

Notes on logical values

- as you might have seen in `read.table(header=T)`, logical values (TRUE, FALSE) can be abbreviated.
- If you want to play a mean prank on someone, write `T <- FALSE; F <- TRUE` in their Rprofile (see `?Startup`).
- Logical (boolean) values F and T internally are often converted to integers 0 and 1, thus `sum(c(T,F,F,T,T,T,F,F,T))` is the number of TRUEs in a vector, `mean` yields the proportion of TRUEs.
- `which(logicalVec)` gives the indices (positions) of TRUE values.
- `Vec[logicalVec]` returns only the values of vec corresponding to TRUE in logicalVec (No need to wrap it into `which`).
- Logical operators: `!, &, |, xor()` (not, and, or, exclusive or)

Notes on logical values

- as you might have seen in `read.table(header=T)`, logical values (TRUE, FALSE) can be abbreviated.
- If you want to play a mean prank on someone, write `T <- FALSE; F <- TRUE` in their Rprofile (see `?Startup`).
- Logical (boolean) values F and T internally are often converted to integers 0 and 1, thus `sum(c(T,F,F,T,T,T,F,F,T))` is the number of TRUEs in a vector, `mean` yields the proportion of TRUEs.
- `which(logicalVec)` gives the indices (positions) of TRUE values.
- `Vec[logicalVec]` returns only the values of vec corresponding to TRUE in logicalVec (No need to wrap it into `which`).
- Logical operators: `!`, `&`, `|`, `xor()` (not, and, or, exclusive or)
- `||` and `&&` for a single logical value.
  In `a || b`, expression `b` will NOT be evaluated if `a` is not TRUE.
  In contrast, when you use `a | b`, expression `b` will be evaluated regardless of outcome of `a`.

Notes on conditional code execution

- `if(c){ex1}` is valid code, thus R doesn't expect else anymore.

Notes on conditional code execution

- `if(c){ex1}` is valid code, thus R doesn't expect else anymore.
- If you execute code line by line (in a script, for example), } and else must be on the same line.

Notes on conditional code execution

- `if(c){ex1}` is valid code, thus R doesn't expect else anymore.
- If you execute code line by line (in a script, for example), } and else must be on the same line.
- Many people consider it good practice to do this in functions as well, but for machine-readability, it is technically fine to write

```
if(cond)
  {
  ex1a
  ex1b
  }
else
  ex2
```

Notes on conditional code execution

- `if(c){ex1}` is valid code, thus R doesn't expect else anymore.
- If you execute code line by line (in a script, for example), } and else must be on the same line.
- Many people consider it good practice to do this in functions as well, but for machine-readability, it is technically fine to write

```
if(cond)
  {
  ex1a
  ex1b
  }
else
  ex2
```

- `if(logicalValue==TRUE) ...` is usually unnecessary, you can write

Notes on conditional code execution

- `if(c){ex1}` is valid code, thus R doesn't expect else anymore.
- If you execute code line by line (in a script, for example), } and else must be on the same line.
- Many people consider it good practice to do this in functions as well, but for machine-readability, it is technically fine to write

```
if(cond)
  {
  ex1a
  ex1b
  }
else
  ex2
```

- `if(logicalValue==TRUE) ...` is usually unnecessary, you can write `if(logicalValue) ...`, but sometimes,

Notes on conditional code execution

- `if(c){ex1}` is valid code, thus R doesn't expect else anymore.
- If you execute code line by line (in a script, for example), } and else must be on the same line.
- Many people consider it good practice to do this in functions as well, but for machine-readability, it is technically fine to write

```
if(cond)
  {
  ex1a
  ex1b
  }
else
  ex2
```

- `if(logicalValue==TRUE) ...` is usually unnecessary, you can write `if(logicalValue) ...`, but sometimes, `if(isTRUE(logicalValue)) ...` helps to deal with NAs.

Actual usage of if else statements

See the `hist` source code:
github.com/wch/r-source -> src / library / graphics / R / hist.R

Actual usage of if else statements

See the `hist` source code:
github.com/wch/r-source -> src / library / graphics / R / hist.R

```
mad
```

Actual usage of if else statements

See the `hist` source code:
github.com/wch/r-source -> src / library / graphics / R / hist.R

```
mad
```

Multiple nested conditionals
`if(a) b else if(c) d else e`
can be avoided with `switch`.

Time to practice

Exercise 25: if else - Conditional code execution

1. `sqrt` returns NaN for the negative values in `v <- -3:5` and warns about it. With a conditional expression, pass 0 instead of negative values to `sqrt`.
2. Construct a statement that checks whether the variable `input <- 4` is a number smaller than 3. Let it write a useful `message` to the console (for both cases). Now test it with `input <- 1.8` and `input <- -17`.
3. Now restrict the correct value of input to *positive* numbers <3, i.e. the number must be <3 AND ≥ 0.
4. BONUS 1: What happens if `input <- "2"` or if `input <- "b"`?
5. BONUS 2: Create a character variable that, depending on the result of `rnorm(1)`, is initiated with "probable" or "unlikely".
6. BONUS 3: `replicate` this experiment 1000 times and examine the result with `table`.
7. BONUS 4: How could you do this with `ifelse`?

Solution for exercise 25.1: if else

```
v <- -3:5
sqrt(v)

## Warning in sqrt(v):  NaNs produced

## [1]    NaN    NaN    NaN 0.0000 1.0000 1.4142 1.7321 2.0000 2.2361
```

```
ifelse( v >= 0, sqrt(v), 0)

## Warning in sqrt(v):  NaNs produced

## [1] 0.0000 0.0000 0.0000 0.0000 1.0000 1.4142 1.7321 2.0000 2.2361
```

```
sqrt(ifelse( v >= 0, v, 0))

## [1] 0.0000 0.0000 0.0000 0.0000 1.0000 1.4142 1.7321 2.0000 2.2361
```

Solution for exercise 25.2: if else

```
input <- 4
if( input >= 3 )
  message("Input was wrong. It should be <3") else
  message("Input OK")

## Input was wrong.  It should be <3

# run it again after
input <- 1.8
input <- -17
```

Solution for exercise 25.3: if else

```
# three different solutions:
if( input >= 3 ) message("Input is > 3") else
if( input < 0 )  message("Input is < 0") else
                 message("Input OK")

if( input >= 3 |  input < 0)
   message("Input outside 0...3") else
   message("Input OK")

if( input > 0  &  input <= 3 )
  message("Input OK") else
  message("Input (",input,") outside 0...3")
```

Solution for exercise 25.4: if else

```r
# characters can be compared to numerics.
# All of the following are TRUE:
"B" > "b"
"b" > 7 # chars are > numbers
"2" > 1
"2" < 3 # but silent conversion is done if possible
```

Solution for exercise 25.5-7: if else

```
result <- if(rnorm(1)>2) "unlikely" else "probable"

result <- replicate(n=1000, expr=
             if(rnorm(1)>2) "unlikely" else "probable")
table(result)

result <- ifelse(rnorm(1000)>2, "unlikely", "probable")
table(result)
```

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
   1. Conditions
   2. For Loops
   3. lapply, Rcpp
   4. Repeat + While Loops
   5. Arrays
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

For loops

Execute a block of code several times, with different input values.

For loops

Execute a block of code several times, with different input values.

Syntax: `for(aRunningVariable in aSequence){ doSomething }`

For loops

Execute a block of code several times, with different input values.

Syntax: `for(aRunningVariable in aSequence){ doSomething }`

Often, i (for index) is used, thus `for(i in 1:n) doThis(i)`

For loops

Execute a block of code several times, with different input values.

Syntax: `for(aRunningVariable in aSequence){ doSomething }`

Often, i (for index) is used, thus `for(i in 1:n) doThis(i)`

Curly brackets are optional if there is one single expression.

For loops

Execute a block of code several times, with different input values.
Syntax: `for(aRunningVariable in aSequence){ doSomething }`
Often, i (for index) is used, thus `for(i in 1:n) doThis(i)`
Curly brackets are optional if there is one single expression.

```
help("for") # needs quotation marks!
```

For loops

Execute a block of code several times, with different input values.
Syntax: `for(aRunningVariable in aSequence){ doSomething }`
Often, i (for index) is used, thus `for(i in 1:n) doThis(i)`
Curly brackets are optional if there is one single expression.

```
help("for") # needs quotation marks!
```

```
print(1:2)
print(1:5)
print(1:9)
```

For loops

Execute a block of code several times, with different input values.
Syntax: `for(aRunningVariable in aSequence){ doSomething }`
Often, i (for index) is used, thus `for(i in 1:n) doThis(i)`
Curly brackets are optional if there is one single expression.

```
help("for") # needs quotation marks!
```

```
print(1:2)
print(1:5)
print(1:9)
```

This is easier and less prone to human errors with:

For loops

Execute a block of code several times, with different input values.
Syntax: `for(aRunningVariable in aSequence){ doSomething }`
Often, i (for index) is used, thus `for(i in 1:n) doThis(i)`
Curly brackets are optional if there is one single expression.

```
help("for") # needs quotation marks!
```

```
print(1:2)
print(1:5)
print(1:9)
```

This is easier and less prone to human errors with:

```
for(i in c(2,5,9) ) { print(1:i) }

## [1] 1 2
## [1] 1 2 3 4 5
## [1] 1 2 3 4 5 6 7 8 9
```

For loops: execute code multiple times I

$$y = f(x, n) = \frac{12.5 * n}{(n-1)!} * \left(\frac{nx}{8}\right)^{(n-1)} * e^{-\frac{nx}{8}}$$

For loops: execute code multiple times I

$$y = f(x, n) = \frac{12.5 * n}{(n-1)!} * \left(\frac{nx}{8}\right)^{(n-1)} * e^{-\frac{nx}{8}}$$

For loops: execute code multiple times I

$$y = f(x, n) = \frac{12.5 * n}{(n-1)!} * \left(\frac{nx}{8}\right)^{(n-1)} * e^{-\frac{nx}{8}}$$

We'll plot the curve for different n values (5,6,7,...,25)

For loops: execute code multiple times II

```
x <- seq(0,25,0.1)
plot(x,x, type="n", ylab="y = f(x,n)")
lines(x, 12.5*5/factorial(5-1)*(x/8*5)^(5-1)*exp(-x/8*5), col="red")
lines(x, 12.5*6/factorial(6-1)*(x/8*6)^(6-1)*exp(-x/8*6), col="red")
```

For loops: execute code multiple times II

```
x <- seq(0,25,0.1)
plot(x,x, type="n", ylab="y = f(x,n)")
lines(x, 12.5*5/factorial(5-1)*(x/8*5)^(5-1)*exp(-x/8*5), col="red")
lines(x, 12.5*6/factorial(6-1)*(x/8*6)^(6-1)*exp(-x/8*6), col="red")
```

For loops: execute code multiple times III

```
x <- seq(0,25,0.1)
plot(x,x, type="n", ylab="y = f(x,n)")
for (n in 5:25)
lines(x, 12.5*n/factorial(n-1)*(x/8*n)^(n-1)*exp(-x/8*n), col="red")
```

For loops: fill a vector

If you follow along with the handout version (one page per slide), please don't view this slide yet. We'll reveal piecewise.

For loops: fill a vector

If you follow along with the handout version (one page per slide), please don't view this slide yet. We'll reveal piecewise.

```
v <- vector(mode="numeric", length=20)   ;  v

##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

For loops: fill a vector

If you follow along with the handout version (one page per slide), please don't view this slide yet. We'll reveal piecewise.

```
v <- vector(mode="numeric", length=20)   ;  v

##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
for(i in 3:17)  {  v[i] <- (i+2)^2  }  # what is now in v?
```

For loops: fill a vector

If you follow along with the handout version (one page per slide), please don't view this slide yet. We'll reveal piecewise.

```
v <- vector(mode="numeric", length=20)   ;  v

##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
for(i in 3:17)  {  v[i] <- (i+2)^2  }  # what is now in v?
```

```
v # the code above was executed once for each i

##  [1]   0   0  25  36  49  64  81 100 121
## [10] 144 169 196 225 256 289 324 361   0
## [19]   0   0
```

For loops: fill a vector

If you follow along with the handout version (one page per slide), please don't view this slide yet. We'll reveal piecewise.

```
v <- vector(mode="numeric", length=20)   ;  v

##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
for(i in 3:17)  {  v[i] <- (i+2)^2  }  # what is now in v?
```

```
v # the code above was executed once for each i

##  [1]   0   0  25  36  49  64  81 100 121
## [10] 144 169 196 225 256 289 324 361   0
## [19]   0   0
```

What objects are now in the global environment workspace?

For loops: fill a vector

If you follow along with the handout version (one page per slide), please don't view this slide yet. We'll reveal piecewise.

```
v <- vector(mode="numeric", length=20)   ;  v

##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
for(i in 3:17)  {  v[i] <- (i+2)^2  }  # what is now in v?
```

```
v # the code above was executed once for each i

##  [1]   0   0  25  36  49  64  81 100 121
## [10] 144 169 196 225 256 289 324 361   0
## [19]   0   0
```

What objects are now in the global environment workspace?
-> `v` (as above) and `i` (with the last assigned value, 17)

For loops: fill a vector

If you follow along with the handout version (one page per slide), please don't view this slide yet. We'll reveal piecewise.

```
v <- vector(mode="numeric", length=20)   ;  v

##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
for(i in 3:17)  {  v[i] <- (i+2)^2  }  # what is now in v?
```

```
v # the code above was executed once for each i

##  [1]   0   0  25  36  49  64  81 100 121
## [10] 144 169 196 225 256 289 324 361   0
## [19]   0   0
```

What objects are now in the global environment workspace?
-> `v` (as above) and `i` (with the last assigned value, 17)

In R, `for` loops are slow. Always try to vectorize (the best option, not always possible) or use `lapply` (less code, easy to parallize).

For loops: multiple graphs

Exercise 26: for loop

1. Read stocks.txt *(rightclick **Raw**, save as)*, so that there are no factors in the data.frame
2. Change the first column type from `char` to `date` with `?as.Date`
3. What do you get with `plot(yourDF[ ,c(1,2)])`? Make it a line graph.
4. With a `for` loop, plot each stock time series, i.e. plot the *i*th column over the first column.
5. BONUS 1: Use good annotations (`main, ylab, xlab`) and turn y axis labels upright (`las`)
6. BONUS 2: Plot all lines in a single graph (set `ylim` in the `plot` command.). Label the lines directly. Use a logarithmic axis, enhanced with `berryFunctions::logAxis`
7. Tough BONUS 3: With `par(mfrow...)`, set up a two by three panel plot. Make the plot margins smaller (`par mar`) and move the axis labels closer to the plots (`mgp`).

For loops: multiple graphs

```
stocks <- read.table("data/stocks.txt", header=T, as.is=T)
stocks$Date <- as.Date(stocks$Date)
par(mfrow=c(2,3), mar=c(2,4,1,1), mgp=c(2.5,0.7,0),oma=c(0,0,2,0),las=1)
for(i in 2:7) plot(stocks[ ,c(1,i)], type="l")
mtext("stocks this decade", line=0, outer=TRUE)
```

logarithmic axis

```
stocks <- read.table("data/stocks.txt", header=TRUE, as.is=TRUE)
stocks$Date <- as.Date(stocks$Date)
companies <- colnames(stocks)[-1]
cols <- c("purple", "orange", "red", "green", "pink", "blue")
names(cols) <- companies     # cols["AMAZON"]

plot(stocks$Date, stocks$APPLE, type="n", log="y", yaxt="n",
     xlim=as.Date(c("2007-01-03","2020-01-01")),
     ylim=range(stocks[,-1], na.rm=TRUE),
     ylab="Stock values, log axis  [USD]", xlab="Date", las=1,
     main="Tech companies rise faster than the car industry")
berryFunctions::logAxis(2)

for(comp in companies)
    lines(stocks$Date, stocks[,comp], col=cols[comp])

#legend("topleft", companies, fill=cols)
berryFunctions::textField(x=tail(stocks[,1],1), labels=companies, adj=0,
                          y=as.numeric(tail(stocks[,-1],1)), col=cols,
                          rrarg=list(bothsame=FALSE), cex=0.8, xpd=TRUE)
```

logarithmic axis

multipanel graph

```
par(mfrow=c(1,2), mar=c(4,4,2,0.1), mgp=c(2.5,0.7,0), cex=0.7, las=1)
for(i in 2:3)
 {
 plot(stocks[ stocks$Date>as.Date("2016-04-01") , c(1,i) ],
      main=colnames(stocks)[i], xaxt="n", type="l", xlab=" ",
      ylab=if(i==2) "Adjusted daily share price  [USD]" else "")
 berryFunctions::monthAxis(1)
 }
```

stocks data from finance.yahoo.com

In case you want to know how I create (and update) the stock dataset:

```
# Download current datasets:
if(!requireNamespace("quantmod")) install.packages("quantmod")
if(!requireNamespace("pbapply"))  install.packages("pbapply")
dummy <- pbapply::pblapply(c("F","VLKAF", "AMZN","AAPL","GOOG","MSFT"),
   function(x) zoo::write.zoo(x=quantmod::getSymbols(x, auto.assign=FALSE)[,6],
                              file=paste0("data/finance/",x,".txt"), col.names=T))

# read single files to R and merge into one file:
stocks <- lapply(dir("data/finance", full=TRUE),
                 read.table, as.is=TRUE, header=TRUE)
stocks <- Reduce(function(...) merge(..., all=T), stocks)

# Get nicer column names:
names <- sapply(strsplit(colnames(stocks), ".", fix=TRUE),"[", 1)
colnames(stocks) <- c(Index="Date", F="FORD", VLKAF="VOLKSWAGEN",
         AMZN="AMAZON", AAPL="APPLE", GOOG="GOOGLE", MSFT="MICROSOFT")[names]

# Save to disc:
write.table(stocks, file="data/stocks.txt", row.names=F, quote=F)
```

`seq_along(n)` is safer than `1:n` in for loops

seq_along(n) is safer than 1:n in for loops

```
do_something <- function(x) if(x<1) stop("x must be >=1, not: ", x) else x
something <- 1:6
```

seq_along(n) is safer than 1:n in for loops

```
do_something <- function(x) if(x<1) stop("x must be >=1, not: ", x) else x
something <- 1:6
```

You'll often see the dangerous code `for(i in 1:n)`:

```
for(i in 1:length(something)) do_something(i) # works with current sth
```

seq_along(n) is safer than 1:n in for loops

```
do_something <- function(x) if(x<1) stop("x must be >=1, not: ", x) else x
something <- 1:6
```

You'll often see the dangerous code `for(i in 1:n)`:

```
for(i in 1:length(something)) do_something(i) # works with current sth
```

Imagine this:

```
something <- which(letters=="4")
for(i in 1:length(something)) do_something(i) # fails! (same code!)

## Error in do_something(i):  x must be >=1, not:  0
```

seq_along(n) is safer than 1:n in for loops

```
do_something <- function(x) if(x<1) stop("x must be >=1, not: ", x) else x
something <- 1:6
```

You'll often see the dangerous code `for(i in 1:n)`:

```
for(i in 1:length(something)) do_something(i) # works with current sth
```

Imagine this:

```
something <- which(letters=="4")
for(i in 1:length(something)) do_something(i) # fails! (same code!)

## Error in do_something(i):  x must be >=1, not:  0
```

Safer to use is:

```
for(i in seq_along(something)) do_something(i)
```

seq_along(n) is safer than 1:n in for loops

```
do_something <- function(x) if(x<1) stop("x must be >=1, not: ", x) else x
something <- 1:6
```

You'll often see the dangerous code `for(i in 1:n)`:

```
for(i in 1:length(something)) do_something(i) # works with current sth
```

Imagine this:

```
something <- which(letters=="4")
for(i in 1:length(something)) do_something(i) # fails! (same code!)

## Error in do_something(i):  x must be >=1, not:  0
```

Safer to use is:

```
for(i in seq_along(something)) do_something(i)
```

Because:

```
1:length(something)     ;   seq_along(something)

## [1] 1 0
## integer(0)
```

for loops: file exercise

Exercise 27: for loops in file creation

We'll write many datasets to disc (and read them back).

1. With `paste0`, print a filename of the structure "mydata_123.txt" using the name, the number and the file ending as inputs. We'll be changing the number later in a loop.
2. Print a data.frame with two columns, each with 10 random numbers: one column from the normal, one from the exponential distribution
3. With `write.table`, write such a table to a file in a subfolder (remember `dir.create`), using the number of rows (e.g. 10) in the filename.
4. BONUS: change the arguments so that row numbers and quotation marks are not printed and tabstops are used for column separation.
5. With a `for`-loop, now write files for different sample sizes, e.g. 10, 20, 50, 100, 500.
6. Using the output of `dir()`, read all the files into a list of data.frames. Remember to first create an empty list.
7. BONUS: name the list elements according to the filenames.
8. BONUS: now replace the whole file reading construct with an `lapply` loop. Celebrate how much nicer your code looks. Check how you can get element names with `sapply(..., simplify=FALSE)`
9. BONUS: With `unlink`, delete the files from this exercise. This function is vectorizable, so there's no need to do this in a for loop!

for loops: file exercise solution

```
dir.create("loopexercise")
for(n in c(10,20,50,100,500))
    write.table(x=data.frame(norm=rnorm(n), exp=rexp(n)),
                file=paste0("loopexercise/randomdata_", n, ".txt"),
                quote=F, row.names=F, sep="\t")

fnames <- dir("loopexercise", full=TRUE)
fcontents <- list()
for(fn in fnames)
   fcontents[[fn]] <- read.table(fn, header=TRUE)

flist <- sapply(dir("loopexercise", full=TRUE),
                read.table, header=TRUE, simplify=FALSE)

all.equal(fcontents, flist) # TRUE

unlink(paste0("loopexercise/randomdata_", c(10,20,50,100,500), ".txt"))
# unlink("loopexercise", recursive=TRUE) # less safe...
```

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
   1. Conditions
   2. For Loops
   3. lapply, Rcpp
   4. Repeat + While Loops
   5. Arrays
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

for -> lapply I: basics

```
files <- dir("rawdata/", pattern="*.csv", full.names=TRUE)
```

for -> lapply I: basics

```
files <- dir("rawdata/", pattern="*.csv", full.names=TRUE)

# bad and slow way to read all files:
ldfs <- list() # initiate empty list
```

for -> lapply I: basics

```
files <- dir("rawdata/", pattern="*.csv", full.names=TRUE)

# bad and slow way to read all files:
ldfs <- list() # initiate empty list
for(i in 1:length(files))
   ldfs[[i]] <- read.csv(files[i], as.is=TRUE)
```

for -> lapply I: basics

```
files <- dir("rawdata/", pattern="*.csv", full.names=TRUE)

# bad and slow way to read all files:
ldfs <- list() # initiate empty list
for(i in 1:length(files))
   ldfs[[i]] <- read.csv(files[i], as.is=TRUE)

# much better way: apply function to each file
ldfs <- lapply(X=files, FUN=read.csv, as.is=TRUE)
```

for -> lapply I: basics

```
files <- dir("rawdata/", pattern="*.csv", full.names=TRUE)

# bad and slow way to read all files:
ldfs <- list() # initiate empty list
for(i in 1:length(files))
   ldfs[[i]] <- read.csv(files[i], as.is=TRUE)

# much better way: apply function to each file
ldfs <- lapply(X=files, FUN=read.csv, as.is=TRUE)


# single data.frame if all files have n columns:
df <- do.call(rbind, ldfs)
```

for -> lapply I: basics

```
files <- dir("rawdata/", pattern="*.csv", full.names=TRUE)

# bad and slow way to read all files:
ldfs <- list() # initiate empty list
for(i in 1:length(files))
   ldfs[[i]] <- read.csv(files[i], as.is=TRUE)

# much better way: apply function to each file
ldfs <- lapply(X=files, FUN=read.csv, as.is=TRUE)


# single data.frame if all files have n columns:
df <- do.call(rbind, ldfs)

# PS: much faster in this example could be
library("data.table") # fread + rbindlist
ldfs <- lapply(X=files, FUN=fread, sep=",")
df <- rbindlist(ldfs)
```

for -> lapply II: progress bar, names, indexing etc

```
ldfs <-   lapply(X=files, FUN=read.csv, as.is=TRUE)
```

for -> lapply II: progress bar, names, indexing etc

```
ldfs <-   lapply(X=files, FUN=read.csv, as.is=TRUE)

# progress bar with remaining time
library("pbapply")
```

for -> lapply II: progress bar, names, indexing etc

```
ldfs <-   lapply(X=files, FUN=read.csv, as.is=TRUE)

# progress bar with remaining time
library("pbapply")
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE)
```

for -> lapply II: progress bar, names, indexing etc

```
ldfs <-   lapply(X=files, FUN=read.csv, as.is=TRUE)

# progress bar with remaining time
library("pbapply")
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE)


# nice additional things:
names(ldfs) <- files
str(ldfs, max.level=1)
ldfs[[2]] # second list element
```

for -> lapply II: progress bar, names, indexing etc

```r
ldfs <-   lapply(X=files, FUN=read.csv, as.is=TRUE)

# progress bar with remaining time
library("pbapply")
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE)


# nice additional things:
names(ldfs) <- files
str(ldfs, max.level=1)
ldfs[[2]] # second list element

# get number of rows for each df:
sapply(ldfs, nrow)
```

for -> lapply II: progress bar, names, indexing etc

```
ldfs <-   lapply(X=files, FUN=read.csv, as.is=TRUE)

# progress bar with remaining time
library("pbapply")
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE)


# nice additional things:
names(ldfs) <- files
str(ldfs, max.level=1)
ldfs[[2]] # second list element

# get number of rows for each df:
sapply(ldfs, nrow)

# get third column / fifth row from each df:
sapply(ldfs, "[",  , j=3)
sapply(ldfs, "[", 5,    )
```

for -> lapply III: parallel on multiple CPU cores

```
library("parallel") # already comes with R
# -> don't try to  install.packages("parallel")
nc <- detectCores()-1 # leave one CPU for xkcd :)
```

for -> lapply III: parallel on multiple CPU cores

```
library("parallel") # already comes with R
# -> don't try to  install.packages("parallel")
nc <- detectCores()-1 # leave one CPU for xkcd :)

# multicore parallel execution: easy on linux
```

for -> lapply III: parallel on multiple CPU cores

```
library("parallel") # already comes with R
# -> don't try to  install.packages("parallel")
nc <- detectCores()-1 # leave one CPU for xkcd :)

# multicore parallel execution: easy on linux
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE)
```

for -> lapply III: parallel on multiple CPU cores

```
library("parallel") # already comes with R
# -> don't try to  install.packages("parallel")
nc <- detectCores()-1 # leave one CPU for xkcd :)

# multicore parallel execution: easy on linux
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE)
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE, cl=nc)
```

for -> lapply III: parallel on multiple CPU cores

```
library("parallel") # already comes with R
# -> don't try to  install.packages("parallel")
nc <- detectCores()-1 # leave one CPU for xkcd :)

# multicore parallel execution: easy on linux
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE)
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE, cl=nc)

# a bit more work on Windows:
```

for -> lapply III: parallel on multiple CPU cores

```
library("parallel") # already comes with R
# -> don't try to  install.packages("parallel")
nc <- detectCores()-1 # leave one CPU for xkcd :)

# multicore parallel execution: easy on linux
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE)
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE, cl=nc)

# a bit more work on Windows:
cl <- makeCluster(nc)
```

for -> lapply III: parallel on multiple CPU cores

```
library("parallel") # already comes with R
# -> don't try to  install.packages("parallel")
nc <- detectCores()-1 # leave one CPU for xkcd :)

# multicore parallel execution: easy on linux
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE)
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE, cl=nc)

# a bit more work on Windows:
cl <- makeCluster(nc)
clusterExport(cl, "files")
```

for -> lapply III: parallel on multiple CPU cores

```
library("parallel") # already comes with R
# -> don't try to  install.packages("parallel")
nc <- detectCores()-1 # leave one CPU for xkcd :)

# multicore parallel execution: easy on linux
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE)
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE, cl=nc)

# a bit more work on Windows:
cl <- makeCluster(nc)
clusterExport(cl, "files")
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE, cl=cl)
```

for -> lapply III: parallel on multiple CPU cores

```
library("parallel") # already comes with R
# -> don't try to  install.packages("parallel")
nc <- detectCores()-1 # leave one CPU for xkcd :)

# multicore parallel execution: easy on linux
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE)
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE, cl=nc)

# a bit more work on Windows:
cl <- makeCluster(nc)
clusterExport(cl, "files")
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE, cl=cl)
stopCluster(cl)
```

for -> lapply III: parallel on multiple CPU cores

```
library("parallel") # already comes with R
# -> don't try to  install.packages("parallel")
nc <- detectCores()-1 # leave one CPU for xkcd :)

# multicore parallel execution: easy on linux
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE)
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE, cl=nc)

# a bit more work on Windows:
cl <- makeCluster(nc)
clusterExport(cl, "files")
ldfs <- pblapply(X=files, FUN=read.csv, as.is=TRUE, cl=cl)
stopCluster(cl)

# template for windows:
berryFunctions::parallelCode()
```

lapply exercise

Exercise 28: lapply

1. Get the average of each element in the built-in dataset `Harman23.cor` with `lapply`.
2. Now use `sapply` instead. What changes?
3. BONUS: Do this again with a `for` loop.
4. Please create `max_n <- function(...)  max(rnorm(1e7))`. Now run this 10 times with an `sapply` loop.
5. Display a progress bar along with it.
6. BONUS: From each element of `Harman74.cor`, get the first 5 values.

lapply exercise solution I

```
lapply(Harman23.cor, mean)

## $cov
## [1] 0.57888
##
## $center
## [1] 0
##
## $n.obs
## [1] 305

sapply(Harman23.cor, mean) # simplified into a vector

##       cov    center     n.obs
##   0.57888   0.00000 305.00000
```

lapply exercise solution II

```
harmeans <- rep(NA,length(Harman23.cor))
for(i in 1:length(Harman23.cor) )
    harmeans[i] <- mean(Harman23.cor[[i]])
names(harmeans) <- names(Harman23.cor)
harmeans

##       cov    center     n.obs
##   0.57888   0.00000 305.00000

max_n <- function(...) max(rnorm(1e5))
pbapply::pbsapply(1:5, max_n)

## [1] 4.6246 4.2047 5.0800 4.2553 4.4011
```

```
lapply(Harman74.cor, "[", 1:5)
```

for -> lapply IVa: parallel on multiple CPU cores

```
library("parallel") # already comes with R
# -> don't try to  install.packages("parallel")
nc <- detectCores()-1

#
```

for -> lapply IVa: parallel on multiple CPU cores

```
library("parallel") # already comes with R
# -> don't try to  install.packages("parallel")
nc <- detectCores()-1

# multicore parallel execution
ldfs <-        lapply(X=files, FUN=read.csv, as.is=TRUE)
ldfs <-      mclapply(X=files, FUN=read.csv, as.is=TRUE,
                      mc.cores=nc) # easy on linux

#
```

for -> lapply IVa: parallel on multiple CPU cores

```
library("parallel") # already comes with R
# -> don't try to  install.packages("parallel")
nc <- detectCores()-1

# multicore parallel execution
ldfs <-        lapply(X=files, FUN=read.csv, as.is=TRUE)
ldfs <-      mclapply(X=files, FUN=read.csv, as.is=TRUE,
                      mc.cores=nc) # easy on linux

# more code needed on windows:
cl <- makePSOCKcluster(nc)
ldfs <- parLapply(cl, X=files, fun=read.csv, as.is=TRUE)
stopCluster(cl)

#
```

for -> lapply IVa: parallel on multiple CPU cores

```
library("parallel") # already comes with R
# -> don't try to  install.packages("parallel")
nc <- detectCores()-1

# multicore parallel execution
ldfs <-        lapply(X=files, FUN=read.csv, as.is=TRUE)
ldfs <-      mclapply(X=files, FUN=read.csv, as.is=TRUE,
                      mc.cores=nc) # easy on linux

# more code needed on windows:
cl <- makePSOCKcluster(nc)
ldfs <- parLapply(cl, X=files, fun=read.csv, as.is=TRUE)
stopCluster(cl)

# sometimes needed before parLapply call:
clusterExport(cl, c("files","otherObjects"))
clusterEvalQ(cl, library("somePackage"))
```

for -> lapply IVb: parallel AND progress bar

```
berryFunctions::parallelCode() # template for windows

library(parallel)
library(pbapply)
cl <- makeCluster( detectCores()-1 )
myfun <- function(...) mean(rnorm(1e6))
clusterExport(cl, "myfun")
dfs1 <- pblapply(X=1:500, FUN=myfun) # 36 sec on 1 CPU core
dfs2 <- pblapply(X=1:500, FUN=myfun, cl=cl) # 10 s on 7
stopCluster(cl)

# selects windows or linux method for you!
# Peter Solymos and Zygmunt Zawadzki are awesome!
```

for -> lapply V: code timing

```
begintime <- Sys.time(); begintime
parLapply(cl, X, fun)
Sys.time() - begintime ; rm(cl, begintime)
```

for -> lapply V: code timing

```
begintime <- Sys.time(); begintime
parLapply(cl, X, fun)
Sys.time() - begintime ; rm(cl, begintime)
```

```
#install.packages("microbenchmark")
library(microbenchmark)
forbad <- function(n)  # fibonacci
  {
  fibvals <- c(1,1)
  for (i in 3:n) fibvals[i] <- fibvals[i-1]+fibvals[i-2]
  fibvals
  }
forgood <- function(n)
  {
  fibvals <- rep(1,n)
  for (i in 3:n) fibvals[i] <- fibvals[i-1]+fibvals[i-2]
  fibvals
  }
mb <- microbenchmark(forbad(2000), forgood(2000))  ;  mb


## Unit: microseconds
##            expr   min     lq    mean median     uq      max neval
##   forbad(2000) 460.0 470.85 3364.31 494.85 518.30 282873.3   100
##  forgood(2000) 178.1 181.70  251.01 183.60 190.65   4011.7   100
```

for -> C++ code: Rcpp

Loops are fast in C++, so outsource that part of your code into C. The package Rcpp will compile it for you and make it available as a normally accessible R function.

for -> C++ code: Rcpp

Loops are fast in C++, so outsource that part of your code into C. The package Rcpp will compile it for you and make it available as a normally accessible R function.
Start learning at http://adv-r.had.co.nz/Rcpp.html

for -> C++ code: Rcpp

Loops are fast in C++, so outsource that part of your code into C. The package Rcpp will compile it for you and make it available as a normally accessible R function.
Start learning at http://adv-r.had.co.nz/Rcpp.html
Speed gain is highly variable, of course, but you might be able to reduce your computing time from 2 hours to 20 seconds!

for -> C++ code: Rcpp

Loops are fast in C++, so outsource that part of your code into C. The package Rcpp will compile it for you and make it available as a normally accessible R function.
Start learning at http://adv-r.had.co.nz/Rcpp.html
Speed gain is highly variable, of course, but you might be able to reduce your computing time from 2 hours to 20 seconds!

To find the slow parts of your code, you perform a process called "profiling":
https://www.r-bloggers.com/profiling-r-code
https://www.rdocumentation.org/packages/utils/topics/Rprof?
https://adv-r.had.co.nz/Profiling.html

for -> C++ code: Rcpp

Loops are fast in C++, so outsource that part of your code into C. The package Rcpp will compile it for you and make it available as a normally accessible R function.
Start learning at http://adv-r.had.co.nz/Rcpp.html
Speed gain is highly variable, of course, but you might be able to reduce your computing time from 2 hours to 20 seconds!

To find the slow parts of your code, you perform a process called "profiling":
https://www.r-bloggers.com/profiling-r-code
https://www.rdocumentation.org/packages/utils/topics/Rprof?
https://adv-r.had.co.nz/Profiling.html
New and very promising:
https://blog.rstudio.org/2016/05/23/profiling-with-rstudio-and-profvis
https://support.rstudio.com/hc/en-us/articles/218221837-Profiling-with-RStudio

For loop: Fibonacci

The Fibonacci sequence consists of the following integers:
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...
The first two values are 1, the following values are always the sum of the previous two elements.

Exercise 29: Fibonacci

1. Create the first 100 numbers of the fibonacci sequence with a for loop
2. BONUS 1: "forget" to create an empty vector of the correct length first, and compare the computational time of the two expressions. Hint: Use `install.packages("microbenchmark"); library(microbenchmark); mb <- microbenchmark(...)`
3. BONUS 2: Can you also do this with sapply?
4. BONUS 3: Find a fast fibonacci creator function online and compare the timings of that as well
5. BONUS 4: With the package `Rcpp`, write the for loop in C++ and compare the speed with the other variants

Fibonacci solution

```
len <- 20
fn <- rep(1, len)
for(i in 3:len)  fn[i] <- fn[i-1] + fn[i-2]
fn

##  [1]    1    1    2    3    5    8   13
##  [8]   21   34   55   89  144  233  377
## [15]  610  987 1597 2584 4181 6765
```

Fibonacci solution II

```
#install.packages("microbenchmark")
library(microbenchmark)
#install.packages("numbers")
suppressMessages(library(numbers))
forbad <- function(n)
  {fibvals <- c(1,1)
  for (i in 3:len) fibvals[i] <- fibvals[i-1]+fibvals[i-2]
  fibvals
  }
forgood <- function(n)
  {fibvals <- rep(1,n)
  for (i in 3:len) fibvals[i] <- fibvals[i-1]+fibvals[i-2]
  fibvals
  }
numfib <- function(n) suppressWarnings(fibonacci(n, seq=T))
applfib <- function(n)
  {fibvals <- rep(1,n)
  sapply(3:n, function(i) {fibvals[i]<<-fibvals[i-1]+fibvals[i-2]})
  }
mb <- microbenchmark(numfib(2000), forbad(2000), forgood(2000), applfib(2000))
mb


## Unit: microseconds
##           expr    min      lq     mean  median      uq     max neval
##   numfib(2000)  324.2  364.95  424.215  409.15  462.15   847.0   100
##   forbad(2000)    6.7    8.00   58.351   11.40   13.30  4588.6   100
##  forgood(2000)    5.9    7.05   48.469    8.80   11.25  3775.3   100
##  applfib(2000) 3715.9 3942.30 4552.700 4066.00 4239.70 26222.0   100
```

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
   1. Conditions
   2. For Loops
   3. lapply, Rcpp
   4. Repeat + While Loops
   5. Arrays
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Repeat loops

Syntax: `repeat {expressions}`

```
next  # jump to next iteration
break # leave the loop
```

Repeat loops

Syntax: `repeat {expressions}`

```
next  # jump to next iteration
break # leave the loop
```

```
i <- 0 # Create object i with the value 0
repeat # keep repeating the following block of code
   {                      # Codeblock begins
   i <- i + 1             # overwrite object i with new number
   if (i < 3)  print(i) # conditional output to console
   if (i == 3)   break  # conditional leaving of loop
   }                      # Codeblock ends

## [1] 1
## [1] 2
```

What is now the value of i?

Repeat loops

Syntax: `repeat {expressions}`

```
next  # jump to next iteration
break # leave the loop
```

```
i <- 0 # Create object i with the value 0
repeat # keep repeating the following block of code
   {                      # Codeblock begins
   i <- i + 1             # overwrite object i with new number
   if (i < 3)  print(i) # conditional output to console
   if (i == 3)   break  # conditional leaving of loop
   }                      # Codeblock ends

## [1] 1
## [1] 2
```

What is now the value of i? i is now 3, because in the last iteration, 1 was added to 2 and the result saved as i.

Repeat loops

Syntax: `repeat {expressions}`

```
next  # jump to next iteration
break # leave the loop
```

```
i <- 0 # Create object i with the value 0
repeat # keep repeating the following block of code
   {                      # Codeblock begins
   i <- i + 1             # overwrite object i with new number
   if (i < 3)  print(i)   # conditional output to console
   if (i == 3)   break    # conditional leaving of loop
   }                      # Codeblock ends

## [1] 1
## [1] 2
```

What is now the value of i? i is now 3, because in the last iteration, 1 was added to 2 and the result saved as i.

```
i <- 0 ; repeat{i <- i+1 ; if(i<3) print(i) ; if(i==3) break}
```

R doesn't care about line breaks in the code.
They are there for us humans. Because we matter more than R.

Repeat loops

Syntax: `repeat {expressions}`

```
next  # jump to next iteration
break # leave the loop
```

```
i <- 0 # Create object i with the value 0
repeat # keep repeating the following block of code
   {                      # Codeblock begins
   i <- i + 1             # overwrite object i with new number
   if (i < 3)  print(i)   # conditional output to console
   if (i == 3)   break    # conditional leaving of loop
   }                      # Codeblock ends

## [1] 1
## [1] 2
```

What is now the value of i? i is now 3, because in the last iteration, 1 was added to 2 and the result saved as i.

```
i <- 0 ; repeat{i <- i+1 ; if(i<3) print(i) ; if(i==3) break}
```

R doesn't care about line breaks in the code.
They are there for us humans. Because we matter more than R. <Blasphemy!>

While loops

Syntax: `while(condition){expressions}`

The following code is executed as long as (while) i is lesser then 3:

```
i <- 0
while(i < 3) i <- i + 1
i

## [1] 3
```

While loops

Syntax: `while(condition){expressions}`

The following code is executed as long as (while) i is lesser then 3:

```
i <- 0
while(i < 3) i <- i + 1
i

## [1] 3
```

`repeat` loops are hardly ever used in R, and `while` loops fairly seldom as well.
`while` loops are helpful when the number of iterations is unknown prior to the task.
To get out of an endlessly repeating `while` loop, hit `ESC` (with the mouse cursor in the console) or the STOP button provided by Rstudio.

Practice while loops

Exercise 30: while loop

1. Why is the character string `folder <- "path/to/file///"` unsuitable for reading in a file?
2. With a while loop, make it suitable. Hints: `substring`, `nchar`
3. BONUS 1: enable your solution to deal with \\ at the end as well. Hint: `%in%`.
4. BONUS 2: Use a regular expression (regexp) instead of `substring`/`nchar`. You'll need `grepl` with `"endswith$"` and `sub`

Solution to exercise 30: while loop

`folder <- "path/to/file///"` would link to empty-named subfolders at the end

```
folder <- "path/to/file///"
while(substring(folder, nchar(folder))=="/")
     folder <- substring(folder, 1, nchar(folder)-1)

while( substring(folder, nchar(folder)) %in% c("/", "\\") )
     folder <- substring(folder, 1, nchar(folder)-1)



# Or, more elegantly, with regular expressions:
while(grepl("/$", folder))  folder <- sub("/$","",folder)
while(grepl("[\\/]$", folder))
     folder <- sub("[\\/]$","",folder)
```

The grand duel (http://www.janko.at/Raetsel/Mathematik/095.a.htm)

Sir Adam and Sir Bart confront each other in a duel with pistols. By good old English custom, the two shoot at each other alternately, until one of them dies. Being gentlemen, they decide that the worse shooter may fire the first shot. Sir Adam has a hit ratio of 30% and Sir Bert one of 40%. What are the chances of survival for each of the two?

Exercise 31: while loop duel

1. BONUS 1: Derive the probabilities mathematically.
2. Simulate the duel numerically. Hint: repeat the experiment `while` both are alive. In each step, let Sir Bert die with a 30% change; if he survives, let Sir Adam die with a 40% change.
3. BONUS 2: Write a function simulating the duel and returning the outcome.
4. Run the simulation 1000 (10k) times and assess Sir Bert's chance of survival.
5. BONUS 3: How many rounds are fought on average? (How is that quantity distributed?)

The grand duel - resolved

```
plot(cumsum(sapply(0:20, function(i) 0.42^i*0.3)), type="l",
     ylim=c(0.26,0.58), main="cumulative death chance ~ round")
lines(cumsum(sapply(0:20, function(i) 0.42^i*0.7*0.4)), col="red")
```

The grand duel - resolved

```
duel_simulation <- function()
{
a <- b <- TRUE # TRUE = 1 = alive
n <- 0 # number of rounds
while(a&b)
  {
  if(  runif(1)<0.3) b <- FALSE # = 0 = dead
  if(b&runif(1)<0.4) a <- FALSE
  n <- n+1
  }
# return simulation result
c(a=a,b=b, n=n)
}
```

The grand duel - resolved

```
simu_res <- replicate(10000, duel_simulation())
# head(t(simu_res), 30)
apply(simu_res, 1, table) # / 10000 *100

## $a
##
##    0    1
## 4800 5200
##
## $b
##
##    0    1
## 5200 4800
##
## $n
##
##    1    2    3    4    5    6    7    8    9   10   11
## 5730 2497 1046  406  197   63   39   12    8    1    1
```

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
   1. Conditions
   2. For Loops
   3. lapply, Rcpp
   4. Repeat + While Loops
   5. Arrays
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Read files in a loop, create an array

Exercise 32: Create an array from files

1. Download the econ.zip file (e.g. with `download.file`), then `unzip` it.
2. Get a vector of filenames in the econ folder with `dir`. Remember to set the `full.names` argument.
3. Using `lapply`, read all the datasets into a list.
4. BONUS: With `basename` and `substr`, use the years in the filename as names of the list elements.
5. Convert each element of the list to a `?matrix`.
6. With `?berryFunctions::l2array`, convert the list to an array.
7. Examine it with `str`

Array creation - solution

```r
download.file(
 "https://github.com/brry/course/raw/master/data/econ.zip",
 destfile="econ.zip")
unzip("econ.zip")
```

```r
files <- dir("data/econ", full.names=TRUE)
econ <- lapply(files, read.table)
names(econ) <- substr(basename(files), 2,5)
econ <- lapply(econ, as.matrix)
econ <- berryFunctions::l2array(econ)
str(econ)

##  num [1:3, 1:3, 1:5] 25.2 2 2.5 321.4 32.1 ...
##  - attr(*, "dimnames")=List of 3
##   ..$ : chr [1:3] "DE" "CH" "GB"
##   ..$ : chr [1:3] "Landwirtschaft" "Industrie" "Dienstleistung"
##   ..$ : chr [1:5] "2012" "2013" "2014" "2015" ...
```

Array subsetting and aggregation

```
some_array[x,y,z, ...]
apply(some_array, MARGIN=c(1,3), FUN=mean)
```

MARGIN denotes the dimensions to keep (i.e. aggregate over the other dimensions)

Exercise 33: subsets and aggregates from arrays

1. Display all the values for Germany (by name). Extract the values in the second economic category (by number). Display a time series of the agricultural branch in Switzerland.
2. Compute the median per category and country (aggregate over time). Compute the maximum of each year.
3. BONUS: Compute the max per year, excluding the industrial sector. If there are many items in the second dimension, how could you automatize the request for a particular entry (like "Industrie") without hard-coding the position? Hint: `str(some_array)` shows a useful attribute you'll need.

Array subsetting - solution

```
econ["DE",,]
econ[,2,]
econ["CH","Landwirtschaft",]

apply(econ, 1:2, median)
apply(econ, 3, max)

apply(econ[,-2,], 3, max)
which(dimnames(econ)[[2]]=="Industrie")
```

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
   1. Functions
   2. Writing R packages
   3. Debugging
   4. Programming exercise
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Functions I

http://r4ds.had.co.nz/functions.html

Functions I

http://r4ds.had.co.nz/functions.html

```
?"function"
```

Functions I

http://r4ds.had.co.nz/functions.html

```
?"function"
```

Syntax:

```
Functionobjectname <- function(argument1, argument2, ...)
                                    {"DoSomething"}
```

Functions I

http://r4ds.had.co.nz/functions.html

`?"function"`

Syntax:

```
Functionobjectname <- function(argument1, argument2, ...)
                                {"DoSomething"}
```

```
myfunct <- function(grappig)
       {plot(grappig, type="l"); return(grappig*7)  }
```

Functions I

http://r4ds.had.co.nz/functions.html

`?"function"`

Syntax:

```
Functionobjectname <- function(argument1, argument2, ...)
                                {"DoSomething"}
```

```
myfunct <- function(grappig)
       {plot(grappig, type="l"); return(grappig*7)  }
```

After `return()`ing, the execution of the function is terminated, so it should only be positioned at the end. It can also be left away, the last instruction ("expression") will then be returned.

Functions II

```
myfunct(   c(5,11,3,7)  )

## [1] 35 77 21 49
```

Functions with more arguments

```
myfunct <- function(x, type="o", ...) plot(x, type=type, ...)
# type="o" is now the default, thus used unless specified
# The ellipsis (...) passes arguments to other functions
myfunct(  c(5,11,3,7) , col="red", las=1)
```

Functions: example I

If you needed to find the zeros of quadratic functions very often, you could use

```
pq <- function(p,q) # y = x^2 + px + q
               {
               w <- sqrt( p^2 / 4 - q )
               c(-p/2-w, -p/2+w)
               } # End of function

pq(3, -12)

## [1] -5.2749  2.2749
```

Functions: example II

```
x <- seq(-10, 10, len=100)
plot(x,  x^2 +3*x -12, type="l", col="red")
abline(h=0, v=0)
```

Exercise: add circles with given radius

```
plot(1:10, asp=1) # aspect ratio y/x of graph range
grid(col="gray") # the next part sould go into a function:
x <- 8 ;  y <- 6 ;  r <-2
p <- seq(0, 2*pi, len=50)
cx <- x+r*cos(p) ;  cy <- y+r*sin(p)
polygon(cx, cy)
```

Time to practice programming

Exercise 34: Writing functions

Write a function that

1. - draws a circle with a certain radius at user-specified locations of an existing plot (see last slide).
2. - uses ellipsis (threedots) to allow the user to customize the appearance
3. - checks all the arguments and gives useful warnings if the wrong type of input is provided
4. - has useful explanations for each argument (documentation)
5. - has readable indentation, spacing and comments explaining the code
6. Now let your neighbor use it without explaining how it is to be used (this should be inferred from the code and comments!)
7. Use your neighbor's function with a vector to draw several circles at once. (unintended use?) What happens?

```r
# Small helper function drawing circles into existing graphics
# Berry Boessenkool, berry-b@gmx.de, 2012
circle <- function(
  x, # x-coordinate of points, numeric value of length 1
  y, # ditto for y
  r, # radius of the circle, in the graphic's units
  locnum=100, # number of points on circle (more means smoother but slower)
  ...) # Further arguments passed to polygon, like col, border, lwd
 {
 # input checking - only one circle can be drawn:
 if(length(x) >1 | length(y) >1 | length(r) >1 | length(locnum) >1)
   {
   warning("Only the first element of the vectors is used.")
   x <- x[1]; y <- y[1]; r <- r[1]; locnum <- locnum[1]
   }
 # input checking - is every value numeric?
 if(!is.numeric(x)) stop("x must be numeric, not ", class(x))
 if(!is.numeric(y)) stop("y must be numeric, not ", class(y))
 if(!is.numeric(r)) stop("r must be numeric, not ", class(r))

 # prepare circle line coordinates:
 cx <- x+r*cos( seq(0,2*pi,len=locnum) )
 cy <- y+r*sin( seq(0,2*pi,len=locnum) )
 polygon(cx, cy, ...) # actually draw it
 }
```

Solution for exercise 34 II: functions

```
plot(1:20, type="n", asp=1, cex=2)
circle(5,5, r=3)
circle(15,10, r=4, locnum=12, col="red", border=4, lwd=3)
```

Solution for exercise 34 III: functions

```
# can not be vectorized:
x <- sample(1:20, 15) ;  y <- sample(1:20, 15) ; r <- runif(20)*4
circle(x,y,r, col=rgb(1,0.5,0,alpha=0.4), border=NA)

## Warning in circle(x, y, r, col = rgb(1, 0.5, 0, alpha = 0.4), border = NA):
Only the first element of the vectors is used.

for(i in 1:15) circle(x[i],y[i],r[i], col=rgb(1,0.5,0,alpha=0.4), border=NA)
```

Coding elegance I

Suppose you have two related vectors, perhaps as a result from `approx`:

```
a:  1   2   3   4   5 ...
b:  1 2 3 4 5 6 7 8 9 ...
```

Coding elegance I

Suppose you have two related vectors, perhaps as a result from `approx`:

```
a:  1   2   3   4   5 ...
b:  1 2 3 4 5 6 7 8 9 ...
```

For any element *i* in `a`, you want to select the corresponding surrounding values of `b`.

Coding elegance I

Suppose you have two related vectors, perhaps as a result from `approx`:

```
a:  1   2   3   4   5 ...
b:  1 2 3 4 5 6 7 8 9 ...
```

For any element *i* in a, you want to select the corresponding surrounding values of b. Example: if i=3, you want to select the elements 4,5 and 6.

Coding elegance I

Suppose you have two related vectors, perhaps as a result from `approx`:

```
a:  1   2   3   4   5 ...
b:  1 2 3 4 5 6 7 8 9 ...
```

For any element *i* in `a`, you want to select the corresponding surrounding values of `b`. Example: if i=3, you want to select the elements 4,5 and 6. However, at the border, you cannot select with negative indices.

Coding elegance I

Suppose you have two related vectors, perhaps as a result from `approx`:

```
a:   1    2    3    4    5 ...
b:   1 2 3 4 5 6 7 8 9 ...
```

For any element *i* in a, you want to select the corresponding surrounding values of b. Example: if i=3, you want to select the elements 4,5 and 6. However, at the border, you cannot select with negative indices.

Exercise 35: Elegant Coding: surrounding indices

1. Write a function using conditionals that performs the task described above.
2. BONUS: If it's not a "one-liner", try to find a different approach.

Coding elegance II

Here is how to write this with conditional statements:

Coding elegance II

Here is how to write this with conditional statements:

```
index_1 <- function(i)
{
if(i == 1)                              1 : 2
else if(i == length(a))  ((i-1)*2) : (i*2-1)
else                     ((i-1)*2) : (i*2)
}
```

Coding elegance II

Here is how to write this with conditional statements:

```
index_1 <- function(i)
{
if(i == 1)                          1 : 2
else if(i == length(a))  ((i-1)*2) : (i*2-1)
else                     ((i-1)*2) : (i*2)
}
```

Here's the elegant way:

```
index_2 <- function(i)
  pmax((i-1)*2, 1) : pmin(i*2, length(a)*2-1)
```

Coding elegance III

Both functions yield the same result:

```
a <- 1:5
index_1(1); index_2(1)

## [1] 1 2
## [1] 1 2

index_1(3); index_2(3)

## [1] 4 5 6
## [1] 4 5 6

index_1(5); index_2(5)

## [1] 8 9
## [1] 8 9
```

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
   1. Functions
   2. Writing R packages
   3. Debugging
   4. Programming exercise
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Why you should write R packages

Why you should write R packages

- Collect your own functions in one place

Why you should write R packages

- Collect your own functions in one place
- Combine functions and documentation in the right way

Why you should write R packages

- Collect your own functions in one place
- Combine functions and documentation in the right way
- Share code with others

Why you should write R packages

- Collect your own functions in one place
- Combine functions and documentation in the right way
- Share code with others
- Make your research reproducible !

Why you should write R packages

- Collect your own functions in one place
- Combine functions and documentation in the right way
- Share code with others
- Make your research reproducible !

Good introductions linked here: github.com/brry/misc

Package development: devtools + github

We'll create a cat package together following the instructions in packdev.R *(rightclick **Raw**, save as)*.

Version control with git:

- slide with Git installation instructions
- github.com, log in, on "+" in top-right select "New repository", initialize with readme!
- On the green field "Code": copy link
- Rstudio - file - new project - version control - git: paste url and create
- `devtools::setup()` instead of `create`, choose option number to Overwrite 'yourPackage.Rproj'
- Work. Commit. Push. Repeat.

Package exercise

Exercise 36: create LSC package (solutions on next 2 slides)

1. Create a package for linear storage cascade functions (lsc)
2. BONUS: Use git from the beginning, commit often!
3. Check the package, correct description file and check again
4. Now add the functions in lsc_functions.zip to the R folder
5. Done? Help your peers.
6. Document the nse function (we'll do this one together)
7. Create datasets within the package (following instructions in data.R)
8. Document the rmse function
9. Check and install the package

Solution for exercise 36: create LSC package

```r
library(devtools)
create("lsc") # or devtools::setup() if using git
document() # calls roxygen2::roxygenise()
check() # will also update documentation
install() # install the package locally
load_all(".") # CTRL + SHIFT + L    - faster than install!
```

Solution for exercise 36: Documenting functions

```
#' @title       Nash-Sutcliffe efficiency
#' @description Nash-Sutcliffe efficiency as a measure of goodness of fit.
#'              Removes incomplete observations with a warning.
#' @return      Single numerical value
#' @export
#' @seealso     \code{\link{rmse}}
#' @references  based on eval.NSeff in RHydro Package
#'              \url{https://r-forge.r-project.org/R/?group_id=411}
#' @examples
#' x <- rnorm(20) ; y <- 2*x + rnorm(20)
#' x[2:4] <- NA
#' nse(x,y)
#'
#' @param obs Numerical vector with observed values
#' @param sim Simulated values (Num vector with same length as \code{obs})
#'
nse <- function(obs, sim)
{
if(!(is.vector(obs) & is.vector(sim))) stop("Input is not a vector.")
if(length(obs) != length(sim)) stop("Vectors are not of equal length.")
#[...]
1 - ( sum((obs - sim)^2) / sum((obs - mean(obs))^2) )
}
```

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
   1. Functions
   2. Writing R packages
   3. Debugging
   4. Programming exercise
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Debugging

- Your code throws an error. You didn't call the mentioned function. Obviously, your code calls some function calling some function calling some function calling [you get the idea] which in the end creates an error. To trace back this path, you can use `traceback()`.

Debugging

- Your code throws an error. You didn't call the mentioned function. Obviously, your code calls some function calling some function calling some function calling [you get the idea] which in the end creates an error. To trace back this path, you can use `traceback()`.
- Now that you know where the error originates from, you set `options(error=recover)`. You run your code again, but this time R waits at the level creating an error. You examine the environment within the function, play around with the objects and internal function code, until the bug has been fixed. You have just debugged a function.

Debugging

- Your code throws an error. You didn't call the mentioned function. Obviously, your code calls some function calling some function calling some function calling [you get the idea] which in the end creates an error. To trace back this path, you can use `traceback()`.
- Now that you know where the error originates from, you set `options(error=recover)`. You run your code again, but this time R waits at the level creating an error. You examine the environment within the function, play around with the objects and internal function code, until the bug has been fixed. You have just debugged a function.
- You want to step into the function you are developing at a specific point. You add `browser()` at that point of the code. You want to go line by line in one specific function. You set `debug(thatFunction)`.

Debugging

- Your code throws an error. You didn't call the mentioned function. Obviously, your code calls some function calling some function calling some function calling [you get the idea] which in the end creates an error. To trace back this path, you can use `traceback()`.
- Now that you know where the error originates from, you set `options(error=recover)`. You run your code again, but this time R waits at the level creating an error. You examine the environment within the function, play around with the objects and internal function code, until the bug has been fixed. You have just debugged a function.
- You want to step into the function you are developing at a specific point. You add `browser()` at that point of the code. You want to go line by line in one specific function. You set `debug(thatFunction)`.
- You want to learn about lexical scoping (Where does R find variables?). http://trestletech.com/2013/04/package-wide-variablescache-in-r-package/ http://adv-r.had.co.nz/Environments.html

Debugging: useful functions

Debugging: useful functions

source("projectFuns.R") execute complete file

Debugging: useful functions

source("projectFuns.R")	execute complete file
traceback()	find error source in sequence of function calls

Debugging: useful functions

source("projectFuns.R")	execute complete file
traceback()	find error source in sequence of function calls
options(warn=2)	warnings to error. default 0

Debugging: useful functions

source("projectFuns.R")	execute complete file
traceback()	find error source in sequence of function calls
options(warn=2)	warnings to error. default 0
browser()	go into function environment: `n`, `s`, `f`, `c`, `Q`

Debugging: useful functions

source("projectFuns.R")	execute complete file
traceback()	find error source in sequence of function calls
options(warn=2)	warnings to error. default 0
browser()	go into function environment: `n`, `s`, `f`, `c`, `Q`
options(error=recover)	open interactive session where error occurred

Debugging: useful functions

source("projectFuns.R")	execute complete file
traceback()	find error source in sequence of function calls
options(warn=2)	warnings to error. default 0
browser()	go into function environment: `n`, `s`, `f`, `c`, `Q`
options(error=recover)	open interactive session where error occurred
debug(funct)	toggle linewise function execution

Debugging: useful functions

source("projectFuns.R")	execute complete file
traceback()	find error source in sequence of function calls
options(warn=2)	warnings to error. default 0
browser()	go into function environment: `n`, `s`, `f`, `c`, `Q`
options(error=recover)	open interactive session where error occurred
debug(funct)	toggle linewise function execution
undebug(funct)	after calling and fixing funct

Debugging: useful functions

source("projectFuns.R")	execute complete file
traceback()	find error source in sequence of function calls
options(warn=2)	warnings to error. default 0
browser()	go into function environment: `n`, `s`, `f`, `c`, `Q`
options(error=recover)	open interactive session where error occurred
debug(funct)	toggle linewise function execution
undebug(funct)	after calling and fixing funct

```
if(length(input)>1) stop("length must be 1, not ", length(input))
```

Debugging: useful functions

source("projectFuns.R")	execute complete file
traceback()	find error source in sequence of function calls
options(warn=2)	warnings to error. default 0
browser()	go into function environment: `n`, `s`, `f`, `c`, `Q`
options(error=recover)	open interactive session where error occurred
debug(funct)	toggle linewise function execution
undebug(funct)	after calling and fixing funct

```
if(length(input)>1) stop("length must be 1, not ", length(input))
```

`stop`: Interrupts function execution and gives error
`warning`: continues but gives warning
`message`: to inform instead of worry the user

Debugging: useful functions

source("projectFuns.R")	execute complete file
traceback()	find error source in sequence of function calls
options(warn=2)	warnings to error. default 0
browser()	go into function environment: `n`, `s`, `f`, `c`, `Q`
options(error=recover)	open interactive session where error occurred
debug(funct)	toggle linewise function execution
undebug(funct)	after calling and fixing funct

```
if(length(input)>1) stop("length must be 1, not ", length(input))
```

`stop`: Interrupts function execution and gives error
`warning`: continues but gives warning
`message`: to inform instead of worry the user

R. Peng (2002): Interactive Debugging Tools in R
D. Murdoch (2010): Debugging in R
H. Wickham (2015): Advanced R: debugging
Example: Pete Werner Blog Post (2013)

Practice debugging, lsc_functions.R *(rightclick **Raw**, save as)*

Exercise 37: Debugging

1. Load your package and the datasets. Correct the functions until `lsc(calib$P, calib$Q, area=1.6)` returns the result below.
2. BONUS: commit each change to git.

Solution for exercise 37: Debugging

- `stupid error you can easily remove` - traceback - find location of error - lsc#73 - just comment it out
- `harder to find but still stupid` - traceback - nse#11 - ditto
- `Error in plot:  need finite 'ylim' value` - debug/browser/options(error=recover) - lsc#105 - NAs in Q - range(Q, na.rm=TRUE) - also in other applicable locations
- `There were 50 or more warnings` - come from rmse being called in optimization - add argument quietNA (or similar) to lsc that is passed to rmse in lsc#79

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
   1. Functions
   2. Writing R packages
   3. Debugging
   4. Programming exercise
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Chapter practice: sample size dependency

Exercise 38: Programming part 1 - functions

1. Write a function (#1) that computes the backtransformed arithmetic average of logtransformed values. It should `stop` with a useful error message if there are `any` negative values in the input vector.
2. Write a second function (#2) taking a vector of values that returns the mean, sd, coefficient of variation and the output of function #1.
3. It should return these values with names as in `c(a=5,b=-3)`.
4. If the average is zero, it should inform the user that the CV cannot be computed. Remember to check only rounded values for equality.
5. Test both functions with `c(88,31,73,21,08,52,66,48)`, with `c(4,7,5,6,4,5,3,NA,6,4,5,6,8,5)` and with `c(0,0,0)`.
6. Write a function (#3) taking **`n`**, **`mean`** and **`sd`** that
   - breaks if **`n`** is not a single value
   - draws **`n`** random numbers from the normal distribution with **`mean`** and **`sd`**
   - returns the output of function #2 of that sample.

Programming practice solutions part 1 - functions 1

backtransformed arithmetic average of logtransformed values

```
logmean <- function(x)
  {
  if(any(x<0)) stop("x has ", sum(x<0), " negative values.")
  lx <- log10(x)
  10^mean(lx)
  }

logmean(c(88,31,73,21,08,52,66,48))

## [1] 39.143

#logmean(c(4,7,5,6,4,5,3,NA,6,4,5,6,8,5))
logmean(c(0,0,0))

## [1] 0
```

Programming practice solutions part 1 - functions 2

Statistical properties of a vector

```
meanvar <- function(vals)
  {
  m <- mean(vals) # average
  s <- sd(vals) # root of variation
  if(round(m)==0) stop("mean is 0, CV cannot be computed")
  c(mean=m, sd=s, CV=s/m, logmean=logmean(vals))
  }

meanvar(c(88,31,73,21,08,52,66,48))

##     mean       sd       CV  logmean
## 48.37500 27.20786  0.56244 39.14336

#meanvar(c(4,7,5,6,4,5,3,NA,6,4,5,6,8,5))
#meanvar(c(0,0,0))
```

Programming practice solutions part 1 - functions 3

Dependency on sample size

```
dep_n <- function(n, mean, sd)
  {
  if(length(n)!=1) stop("n must be a single value, not ",
                        length(n))
  # draw n numbers randomly from normal distribution:
  v <- rnorm(n, mean=mean, sd)
  meanvar(v)
  }

dep_n(20, 70,5)

##      mean        sd        CV   logmean
## 70.959600  6.563142  0.092491 70.658167

#
```

Chapter practice: sample size dependency

Exercise 39: Programming part 2 - debugging

1. Test function #3 with `n=20:30, mean=70,sd=5`
2. What happens if you use it with `n=NA, mean=70,sd=5`?
3. After you call it with `n=20, mean=-70,sd=5`, run `traceback()` Explain the result.
4. What is the result of `ls.str()`?
5. Set you upper level function #3 to `debug`ging mode. Now call it again. What happens? What is now the result of `ls.str()`?

- With enter, you can now execute the function line by line and observe how objects (variables) in functions change over execution time, thus making it easier to find the source of an error.
- Don't forget to `undebug` your function again.
- If you only want to step into the function at a certain point, write `browser()` at that position, redefine the function and call it again.

Programming practice solutions part 2 - debugging

```
dep_n(20, -70,5)

## Error in logmean(vals):  x has 20 negative values.
```

```
traceback()
```

```
4:  stop("x has ", sum(x < 0), " negative values.") at #3
3:  logmean(vals) at #6
2:  meanvar(v) at #5
1:  dep_n(20, -70, 5)
```

In debugging mode, you step into the function when it is called (visible by `Browse[2]>` in the console). `ls.str()` now returns the structures of the objects defined within the function environment.

Chapter practice: sample size dependency

Exercise 40: Programming part 3 - loops for computation

For this task, use `n <- rep(1:50, each=100)`

1. Create an empty `data.frame` with the column names according to the output of function #3.
2. With a for loop, fill each row with the result of the n-corresponding call to function #3.
3. Obtain the computing time by either wrapping it in `system.time` or calling `Sys.time()` before and afterwards and computing the `difftime`.
4. Time the usage of `sapply` for the same task. How many times is it faster? What does that imply for a 5-hour for loop computation?
5. BONUS: Get uncertainties on the computing time estimation with `microbenchmark` (You might want to reduce the length of n for that).

Programming practice solutions part 3 - loops computation

```
n <- rep(1:50, each=100)
results_for <- data.frame(mean=NA, sd=NA, CV=NA, logmean=NA)
system.time( for(i in 1:length(n))
  {results_for[i, ] <- dep_n(n[i], 70,5)})

##    user  system elapsed
##    2.50    0.02    2.58

tail(results_for, 5)

##        mean     sd       CV logmean
## 4996 70.076 4.8519 0.069238  69.909
## 4997 70.303 4.7884 0.068111  70.144
## 4998 69.700 5.4201 0.077764  69.500
## 4999 69.354 4.7802 0.068925  69.192
## 5000 68.476 4.4889 0.065554  68.330
```

Programming practice solutions part 3 - loops computation

```
system.time(res_lapply <- sapply(n, dep_n, mean=70,sd=5)  )

##    user  system elapsed
##    0.25    0.01    0.40
```

In this case with (only) 5000 iterations, lapply is ca 7 times faster than the badly initiated for loop. 5 hrs of computation would reduce to 40 minutes. Notice you get similar to lapply speed results for properly initiated data.frames with the correct dimension.
For more speed hints, read win-vector.com efficient accumulation in R.
Remember: your code is much cleaner with lapply:

```
results <- matrix(NA, ncol=4, nrow=length(n2))
for(i in 1:length(n2)) results[i, ] <- dep_n(n2[i], 70,5)

results <- sapply(n2, dep_n, mean=70,sd=5)
```

Programming practice solutions part 3 - loops computation

```
n2 <- rep(1:50, each=10)

dep_n_for <- function()
 {results <- data.frame(mean=NA, sd=NA, CV=NA, logmean=NA)
  for(i in 1:length(n2)) results[i, ] <- dep_n(n2[i], 70,5)}

dep_n_for_good <- function()
 {results <- matrix(NA, ncol=4, nrow=length(n2))
  for(i in 1:length(n2)) results[i, ] <- dep_n(n2[i], 70,5)}

dep_n_sapply <- function()
{results <- sapply(n2, dep_n, mean=70,sd=5)}

mb <- microbenchmark(dep_n_for(), dep_n_for_good(),
                     dep_n_sapply())
save(mb, file="data/mb.Rdata")
```

Programming practice solutions part 3 - loops computation

```
load("data/mb.Rdata"); library(microbenchmark)
boxplot(mb, yaxt="n"); berryFunctions::logAxis(2)
```

Chapter practice: sample size dependency

Exercise 41: Programming part 4 - loops for plotting

1. Plot the mean estimate over sample size, using fully colored but semitransparent dots (see `rgb` or `berryFunctions::addAlpha`).
2. Add a nicely visible horizontal line at the population mean. (we drew random samples from some defined distribution, which is the general population whose parameters are usually unknown in real life).
3. BONUS: add lines limiting the 90% confidence region.
4. Set up a multipanel plot to automatically hold a figure for each of the rows of the previous results (mean, sd, CV, logmean). To automatically dimension the panels, you could use `berryFunctions::panelDim(n)`
5. With a for loop, fill each of the panels and add nice annotations.

Programming practice solutions part 4 - loops plotting

```
transp <- rgb(0,0,0, alpha=0.1)
plot(n, res_lapply["mean", ], pch=16, col=transp, cex=0.6)
abline(h=70, col="blue", lwd=2)
```

Programming practice solutions part 4 - loops plotting

```
conf <- tapply(res_lapply["mean", ], n, quantile, probs=c(0.05, 0.95))
conf <- sapply(conf, I)
conf_x <- as.numeric(colnames(conf))
for(i in 1:2) lines(conf_x, conf[i, ], col="red", lwd=2)
```

Programming practice solutions part 4 - loops plotting

```
library(berryFunctions) # for panelDim

par(mfrow=panelDim(nrow(res_lapply)), mar=c(2,4,1,1),
    mgp=c(2.5,0.7,0), las=1 )

truevals <- c(mean=70, sd=5, CV=5/70, logmean=70)
for(nm in rownames(res_lapply))
  {
  plot(n, res_lapply[nm, ], ylab=nm, pch=16, col=transp)
  abline(h=truevals[nm], col="blue", lwd=2)
  if(nm=="sd") points(60,2, col="red")
  }
```

Programming practice solutions part 4 - loops plotting

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
   1. Exploratory Data Analysis (EDA)
   2. Normal distribution
   3. Other distribution functions
   4. Beta distribution
   5. Extreme Value Statistics
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

A quick example I

```
grades <- c(3.0, 2.3, 5, 1.3, 4, 1.7)
grades          # Show the object

## [1] 3.0 2.3 5.0 1.3 4.0 1.7

mode(grades)    # numeric

## [1] "numeric"

sum(grades)     # sum of all values

## [1] 17.3

diff(grades)    # difference from one value to the next

## [1] -0.7  2.7 -3.7  2.7 -2.3
```

```r
mean(grades)    # Average

## [1] 2.8833

median(grades) # median*

## [1] 2.65

var(grades)     # variance

## [1] 1.9977

sd(grades)      # Standard deviation

## [1] 1.4134
```

* If you sort all the numbers ascendingly, the median is in the middle. Thus, it is not as sensitive to outliers as the mean. We'll get back to this later on.

The dataset we'll be looking at

sources: mirlab.org, desirableplants.com, 3.bp.blogspot.com

iris data set

```r
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Exercise 42: EDA - Exploratory Data Analysis

1. Look at `iris` with `summary`.
2. Assign `iris$Sepal.Length` to an object with a short name.
3. Calculate `min` and `max`, the `median` and the median absolute deviation (`mad`), the quartiles and octiles (via `quantile`), the fourthspread (`IQR`) and `diff(range)` of the iris values.
4. With `par(mfrow=c(2,1))`, set the graphic device to contain two graphics.
5. Draw a horizontal `boxplot` in the upper window and compare it with the statistics calculated above. Interpret them.
6. What happens when the argument `notch` is set to true?
7. Draw a `hist`ogram with the bars filled with orange color, and experiment with the number of breaks.
8. BONUS: What are the (dis-)advantages of boxplots and histograms?

Mean vs. Median etc.

mathwithbaddrawings.com/2016/07/13/why-not-to-trust-statistics

Mean vs. Median etc.

mathwithbaddrawings.com/2016/07/13/why-not-to-trust-statistics

Similar boxplots...

... different distributions!

Random numbers

R has a very good random number generator. If you want to set a starting point for it, you can use `set.seed(n)`, with n being some number you like. After R has set the seed, it will always generate the same "random" numbers. This creates reproducible numbers.

Exercise 43: Random numbers

1. With `sample`, put the numbers 5 to 15 in random order. Out of the possible numbers 1:3, draw 25 realizations.
2. Via `?Distributions`, find out how to generate 200 random numbers from the exponential distribution and look at those with a boxplot and a histogram (BONUS: look at `ecdf` and `density`).

Solution for exercise 43: random numbers

```
sample(5:15)  ;  sample(1:3, 25, replace=TRUE)  ;  r <- rexp(200)
par(mfrow=c(2,1), mar=c(2,2,0,0))
hist(r, breaks=20, col="green4")
boxplot(r, horizontal=T, notch=T, col="green4")
lines(density(r)$x, density(r)$y +par("usr")[3], lwd=3)
lines(0:50/10, dexp(0:50/10) +par("usr")[3], col="red")
```

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
   1. Exploratory Data Analysis (EDA)
   2. Normal distribution
   3. Other distribution functions
   4. Beta distribution
   5. Extreme Value Statistics
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

normal distribution: data

```
url <- rdwd::selectDWD("Potsdam", res="annual", var="kl", per="h")
clim <- rdwd::dataDWD(url)
clim$Year <- as.numeric(substr(clim$MESS_DATUM_BEGINN, 1, 4))
plot(clim$Year, clim$JA_RR, type="l", col="blue", lwd=3,
     xlab="Year", ylab="Precipitation sum  [mm]",
     main="Annual rainfal in Potsdam appears trendless")
```

normal distribution: empirical histogram

```
hist(clim$JA_RR, breaks=20, col="azure3", main="Histogram Rainfall",
     ylab="Number of values in rainfall bin",
     xlab="Annual rainfall sum Potsdam 1893:2018")
```

normal distribution: density histogram

normal distribution: density histogram

normal distribution: density histogram

normal distribution: density histogram

normal distribution: density histogram

normal distribution: density histogram

normal distribution: code

```
par(mgp=c(3,0.7,0), las=1)
hist(clim$JA_RR, breaks=20, col="azure3",
     main="Probability density, total histogram area=1",
     xlab="Annual rainfall sum Potsdam 1893:2018", freq=FALSE)
abline(v=mean(clim$JA_RR), col="red", lwd=5)
textField(650, 0.005, paste0("mean = ", round(mean(clim$JA_RR),0)), col="red")
musd <- function(x,f=1) mean(x,na.rm=TRUE)+c(-f,f)*sd(x, na.rm=TRUE)
segments(musd(clim$JA_RR,1), rep(-1,2), y1=rep(0.0025,2), col="red", lwd=5)
textField(690, 0.004, paste0("sd = ", round(sd(clim$JA_RR),0)), col="red")
textField(589, 0.002, "70% of values between\nmu +- 1 sd")
segments(musd(clim$JA_RR,2), rep(-1,2), y1=rep(0.0006,2), col="red", lwd=5)
textField(589, 0.0007, "95% of values between\nmu +- 2 sd")
lines(200:900, dnorm(200:900, mean(clim$JA_RR), sd(clim$JA_RR)), col="red",lwd=3)
```

Distributions: normal distribution I

Distributions: normal distribution II

Distributions: normal distribution II

Distributions: normal distribution II

Distributions: normal distribution II

`pnorm(q=615, mean=600, sd=10)` = 93% of the values of a normal distribution (ND) with $\mu = 600$ and $\sigma = 10$ are smaller than 615.

`qnorm(p=0.4, mean=600, sd=10)` = 597 marks the point with 40% of values below it.

Get familiar with the Gaussian distribution

Exercise 44: normal distribution

In 1981, Fries and Crapo estimated future life expectancy to be 85 years with a standard deviation of 4.5 years.

Fries, J., Crapo, L.: Vitality and Aging: Implications of the Rectangular Curve. Freeman & Co Publishers, San Francisco (1981)

1. Plot the distribution density curve
2. Plot it again using `normPlot` in the package `berryFunctions`.
3. What proportion of people would die before age 75?
   Is that a realistic likelihood?
4. How many Germans (pop ~80 mio) would exceed 90 years?
5. In which symmetric age range would 95% of the people die?
   Give the rough estimate and the exact number.
6. How likely is it to die at age 85.3?
7. What are the biggest problems with this task?

Solution for exercise 44: Normal distribution I

Examples taken from: Wolfgang Tschirk (2014): Statistik: Klassisch oder Bayes - Zwei Wege im Vergleich

```
plot(70:100, dnorm(70:100, 85, 4.5), type="l")
berryFunctions::normPlot(85, 4.5)
pnorm(75, 85, 4.5)*100 #  1.3%
(1-pnorm(90, 85, 4.5))*80e6   # 10.7 mio
85-4.5*2; 85+4.5*2 # 76, 94
qnorm(c(0.025, 0.975), 85, 4.5) # 76.2 - 93.8
```

The probability to die at age 85.3 is 0 percent, as there are infinite possibilities in a continuous distribution. (You could also die at 85.331761 years). That's why we look at probability "density" in intervals.
But: we don't know if life expectancy is normally distributed. We also don't know whether it is parametrized correctly.

Solution for exercise 44: Normal distribution II

```
## [1] 1.3134
## [1] 10660821
## [1] 76.18 93.82
```

Test for normality of a distribution

```
data <- rnorm(1000, mean=97, sd=8.9)
shapiro.test(data)
ks.test(data, "pnorm", mean(data), sd(data))
# if p > 0.05: accept Nullhypothesis
# (that data are normally distributed.)
```

More on slide 372 et sequentes.

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
   1. Exploratory Data Analysis (EDA)
   2. Normal distribution
   3. Other distribution functions
   4. Beta distribution
   5. Extreme Value Statistics
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Get familiar with the discrete Binomial distribution

Exercise 45: binomial distribution

149'291 of 279'371 students in Austria are female. Statistik Austria: Studierende in OEsterreich im Wintersemester 2009/10. Statistik Austria, Wien (2010). This is a population for which we know the actual proportion, which happens rarely in inferential statistics. 100 students are randomly chosen.

1. Guess how likely it is that the majority (at least half) of those are female.
2. Plot the probability for each reasonable possible outcome using `dbinom`. (This is the actual probability, not probability density, for each discrete number of females in a group of 100 randomly chosen students). Now guess again.
3. Actually calculate the probability.
4. At which proportion of females would you accuse the experimenter of selection bias (or cheating)?

Solution for exercise 45: Binomial distribution

```
prop_f <- 149291/279371 # Proportion of females among students in Austria
plot(30:70, dbinom(x=30:70, size=100, prob=prop_f)*100,
     ylab="Probability for n females\nin a sample of 100 stundents  [%]")
HDI <- qbinom(p=c(0.025, 0.975), size=100, prob=prop_f)
abline(v=c(49.5, prop_f*100, HDI), col=c("purple","grey","red","red"))
1-pbinom(q=49.5, size=100, prob=prop_f)   ;  HDI


## [1] 0.78528
## [1] 44 63
```

In 78.5% of samples, a majority should be female. Improper sampling is assumed if the probability of the result happening by chance is under e.g. 5%, in this case ≤ 44 or ≥ 63 females (two-sided confidence interval).

Other distributions: density

Other distributions: cumulated

Overview of distributions in R

prefix	meaning	usage
d	**d**ensity	calculate probability (density) for value x
p	cumulated **p**robability	area under density curve up until value x
q	**q**uantile	reciprocal to p: calculate value x for a given cumulated probability
r	**r**andom	draw random numbers from distribution

Overview of distributions in R

prefix	meaning	usage
d	**d**ensity	calculate probability (density) for value x
p	cumulated **p**robability	area under density curve up until value x
q	**q**uantile	reciprocal to p: calculate value x for a given cumulated probability
r	**r**andom	draw random numbers from distribution

distribution	parameters	type
binom	size, prob	discrete
pois	lambda	
norm	mean, sd	continuous
exp	rate	
beta	shape1, shape2	

Exercise 46: classical statistical analysis

1. Calculate `mean`, `var`, `sd` and standard error of the mean (SEM $= \frac{\sigma}{\sqrt{n}}$) of the `iris` sepal length data and interpret these values.
2. Visualize the theoretical distribution of the dataset. Use a sequence of values and calculate `pnorm` and `dnorm` on it with mean and sd according to the previous task.
3. Compare this with the actual distribution of the sample. First draw a histogram with `freq=FALSE`, then add the theoretical distribution with `lines`.
4. If this distribution were from the general and complete "population" of iris sepal lengths, we could say: 20% of the sepal lengths are smaller than a certain value. Use `qnorm` to find out this value. What is the limit of this approach?
5. BONUS: What are the (dis-)advantages of mean and median?

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
   1. Exploratory Data Analysis (EDA)
   2. Normal distribution
   3. Other distribution functions
   4. Beta distribution
   5. Extreme Value Statistics
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Beta distribution: motivating example

Beta distribution: motivating example

Imagine you are the scientific advisor to the government of India.

Beta distribution: motivating example

Imagine you are the scientific advisor to the government of India.
It has recently been found that some of the millions (yes, millions) of wells contain dangerous levels of arsenic.

Beta distribution: motivating example

Imagine you are the scientific advisor to the government of India.
It has recently been found that some of the millions (yes, millions) of wells contain dangerous levels of arsenic.
If more than 25% of the wells are affected, the government will pass a law constraining well construction.

Beta distribution: motivating example

Imagine you are the scientific advisor to the government of India.
It has recently been found that some of the millions (yes, millions) of wells contain dangerous levels of arsenic.
If more than 25% of the wells are affected, the government will pass a law constraining well construction.
You sample 10 wells and find that 2 surpass the safety limit of arsenic concentration.

Beta distribution: motivating example

Imagine you are the scientific advisor to the government of India.
It has recently been found that some of the millions (yes, millions) of wells contain dangerous levels of arsenic.
If more than 25% of the wells are affected, the government will pass a law constraining well construction.
You sample 10 wells and find that 2 surpass the safety limit of arsenic concentration. What will you do?

Beta distribution: motivating example

Imagine you are the scientific advisor to the government of India.
It has recently been found that some of the millions (yes, millions) of wells contain dangerous levels of arsenic.
If more than 25% of the wells are affected, the government will pass a law constraining well construction.
You sample 10 wells and find that 2 surpass the safety limit of arsenic concentration. What will you do?
To graph the uncertainty, you want to use the beta distribution.

Beta distribution: motivating example

Imagine you are the scientific advisor to the government of India.
It has recently been found that some of the millions (yes, millions) of wells contain dangerous levels of arsenic.
If more than 25% of the wells are affected, the government will pass a law constraining well construction.
You sample 10 wells and find that 2 surpass the safety limit of arsenic concentration. What will you do?
To graph the uncertainty, you want to use the beta distribution.
You remember (from this class) that for a binomial trial, the two parameters can be defined as:

Beta distribution: motivating example

Imagine you are the scientific advisor to the government of India.
It has recently been found that some of the millions (yes, millions) of wells contain dangerous levels of arsenic.
If more than 25% of the wells are affected, the government will pass a law constraining well construction.
You sample 10 wells and find that 2 surpass the safety limit of arsenic concentration. What will you do?
To graph the uncertainty, you want to use the beta distribution.
You remember (from this class) that for a binomial trial, the two parameters can be defined as:
$\alpha - 1 =$ number of successes

Beta distribution: motivating example

Imagine you are the scientific advisor to the government of India.
It has recently been found that some of the millions (yes, millions) of wells contain dangerous levels of arsenic.
If more than 25% of the wells are affected, the government will pass a law constraining well construction.
You sample 10 wells and find that 2 surpass the safety limit of arsenic concentration. What will you do?
To graph the uncertainty, you want to use the beta distribution.
You remember (from this class) that for a binomial trial, the two parameters can be defined as:
$\alpha - 1 =$ number of successes
$\beta - 1 =$ number of failures

Beta distribution: motivating example

Imagine you are the scientific advisor to the government of India.
It has recently been found that some of the millions (yes, millions) of wells contain dangerous levels of arsenic.
If more than 25% of the wells are affected, the government will pass a law constraining well construction.
You sample 10 wells and find that 2 surpass the safety limit of arsenic concentration. What will you do?
To graph the uncertainty, you want to use the beta distribution.
You remember (from this class) that for a binomial trial, the two parameters can be defined as:
$\alpha - 1 =$ number of successes
$\beta - 1 =$ number of failures
What are the parameters in this case?

Beta distribution: motivating example

Imagine you are the scientific advisor to the government of India.
It has recently been found that some of the millions (yes, millions) of wells contain dangerous levels of arsenic.
If more than 25% of the wells are affected, the government will pass a law constraining well construction.
You sample 10 wells and find that 2 surpass the safety limit of arsenic concentration. What will you do?
To graph the uncertainty, you want to use the beta distribution.
You remember (from this class) that for a binomial trial, the two parameters can be defined as:
$\alpha - 1 =$ number of successes
$\beta - 1 =$ number of failures
What are the parameters in this case?

$\alpha =$

Beta distribution: motivating example

Imagine you are the scientific advisor to the government of India.
It has recently been found that some of the millions (yes, millions) of wells contain dangerous levels of arsenic.
If more than 25% of the wells are affected, the government will pass a law constraining well construction.
You sample 10 wells and find that 2 surpass the safety limit of arsenic concentration. What will you do?
To graph the uncertainty, you want to use the beta distribution.
You remember (from this class) that for a binomial trial, the two parameters can be defined as:
$\alpha - 1 =$ number of successes
$\beta - 1 =$ number of failures
What are the parameters in this case?

$$\alpha = 2 + 1 =$$

Beta distribution: motivating example

Imagine you are the scientific advisor to the government of India.
It has recently been found that some of the millions (yes, millions) of wells contain dangerous levels of arsenic.
If more than 25% of the wells are affected, the government will pass a law constraining well construction.
You sample 10 wells and find that 2 surpass the safety limit of arsenic concentration. What will you do?
To graph the uncertainty, you want to use the beta distribution.
You remember (from this class) that for a binomial trial, the two parameters can be defined as:
$\alpha - 1 =$ number of successes
$\beta - 1 =$ number of failures
What are the parameters in this case?

$$\alpha = 2 + 1 = 3$$

Beta distribution: motivating example

Imagine you are the scientific advisor to the government of India.
It has recently been found that some of the millions (yes, millions) of wells contain dangerous levels of arsenic.
If more than 25% of the wells are affected, the government will pass a law constraining well construction.
You sample 10 wells and find that 2 surpass the safety limit of arsenic concentration. What will you do?
To graph the uncertainty, you want to use the beta distribution.
You remember (from this class) that for a binomial trial, the two parameters can be defined as:
$\alpha - 1 =$ number of successes
$\beta - 1 =$ number of failures
What are the parameters in this case?

$$\alpha = 2 + 1 = 3$$
$$\beta =$$

Beta distribution: motivating example

Imagine you are the scientific advisor to the government of India.
It has recently been found that some of the millions (yes, millions) of wells contain dangerous levels of arsenic.
If more than 25% of the wells are affected, the government will pass a law constraining well construction.
You sample 10 wells and find that 2 surpass the safety limit of arsenic concentration. What will you do?
To graph the uncertainty, you want to use the beta distribution.
You remember (from this class) that for a binomial trial, the two parameters can be defined as:
$\alpha - 1 =$ number of successes
$\beta - 1 =$ number of failures
What are the parameters in this case?

$$\alpha = 2 + 1 = 3$$
$$\beta = 10 - 2 + 1 =$$

Beta distribution: motivating example

Imagine you are the scientific advisor to the government of India.
It has recently been found that some of the millions (yes, millions) of wells contain dangerous levels of arsenic.
If more than 25% of the wells are affected, the government will pass a law constraining well construction.
You sample 10 wells and find that 2 surpass the safety limit of arsenic concentration. What will you do?
To graph the uncertainty, you want to use the beta distribution.
You remember (from this class) that for a binomial trial, the two parameters can be defined as:
$\alpha - 1 =$ number of successes
$\beta - 1 =$ number of failures
What are the parameters in this case?

$$\alpha = 2 + 1 = 3$$
$$\beta = 10 - 2 + 1 = 9$$

Beta distribution: some theory + density plot

Beta distribution: some theory + density plot

- Continuous, values between 0 and 1

Beta distribution: some theory + density plot

- Continuous, values between 0 and 1
- Suitable for proportions and probabilities

Beta distribution: some theory + density plot

- Continuous, values between 0 and 1
- Suitable for proportions and probabilities
- Very versatile, different shapes

Beta distribution: some theory + density plot

- Continuous, values between 0 and 1
- Suitable for proportions and probabilities
- Very versatile, different shapes
- Used extensively in Bayesian statistics, see e.g. Liu and Kong (2015).

Beta distribution: some theory + density plot

- Continuous, values between 0 and 1
- Suitable for proportions and probabilities
- Very versatile, different shapes
- Used extensively in Bayesian statistics, see e.g. Liu and Kong (2015).
- Two parameters: a=alpha=`shape1` and b=beta=`shape2`.

Beta distribution: some theory + density plot

- Continuous, values between 0 and 1
- Suitable for proportions and probabilities
- Very versatile, different shapes
- Used extensively in Bayesian statistics, see e.g. Liu and Kong (2015).
- Two parameters: a=alpha=`shape1` and b=beta=`shape2`.

Exercise 47: Beta distribution density

1. Plot the beta distribution density (`dbeta`) for alpa=3 and beta=9.
2. If this describes the true distribution of contaminated well proportions across subregions of India, how likely is it to have a sample with 25% or more of the wells contaminated?
3. BONUS: Plot the distribution with `polygon`.

Two hints:

Beta distribution: some theory + density plot

- Continuous, values between 0 and 1
- Suitable for proportions and probabilities
- Very versatile, different shapes
- Used extensively in Bayesian statistics, see e.g. Liu and Kong (2015).
- Two parameters: a=alpha=`shape1` and b=beta=`shape2`.

Exercise 47: Beta distribution density

1. Plot the beta distribution density (`dbeta`) for alpa=3 and beta=9.
2. If this describes the true distribution of contaminated well proportions across subregions of India, how likely is it to have a sample with 25% or more of the wells contaminated?
3. BONUS: Plot the distribution with `polygon`.

Two hints:

Hint 1: Create a vector from 0 to 1 with, say, 200 values.

Beta distribution: some theory + density plot

- Continuous, values between 0 and 1
- Suitable for proportions and probabilities
- Very versatile, different shapes
- Used extensively in Bayesian statistics, see e.g. Liu and Kong (2015).
- Two parameters: a=alpha=`shape1` and b=beta=`shape2`.

Exercise 47: Beta distribution density

1. Plot the beta distribution density (`dbeta`) for alpa=3 and beta=9.
2. If this describes the true distribution of contaminated well proportions across subregions of India, how likely is it to have a sample with 25% or more of the wells contaminated?
3. BONUS: Plot the distribution with `polygon`.

Two hints:

Hint 1: Create a vector from 0 to 1 with, say, 200 values.

Hint 2: Plot `dbeta(x)` over x using `type="l"`.

Solution to exercise 47: Beta density

```
proportion <- seq(0,1, len=200)
plot(proportion, dbeta(proportion, 3,9), type="l")
segments(x0=0.25, y0=-1, y1=3.097, col="red")
```

Solution to exercise 47: Beta density

```
proportion <- seq(0,1, len=200)
plot(proportion, dbeta(proportion, 3,9), type="l")
segments(x0=0.25, y0=-1, y1=3.097, col="red")
```

```
1-pbeta(0.25, 3,9)

## [1] 0.4552
```

Beta parameters

Beta parameters

Large α: HDI (Highest Density Interval) more to the *left/right*?

Beta parameters

Large α: HDI (Highest Density Interval) more to the right

Beta parameters

Large α: HDI (Highest Density Interval) more to the right
Large β: HDI rather left, lower values

Beta parameters

Large α: HDI (Highest Density Interval) more to the right
Large β: HDI rather left, lower values
Both large:

Beta parameters

Large α: HDI (Highest Density Interval) more to the right
Large β: HDI rather left, lower values
Both large: (*more samples*)

Beta parameters

Large α: HDI (Highest Density Interval) more to the right
Large β: HDI rather left, lower values
Both large: (*more samples*) narrow HDI

Beta parameters

Large α: HDI (Highest Density Interval) more to the right
Large β: HDI rather left, lower values
Both large: (*more samples*) narrow HDI

```
install.packages("berryFunctions")
```

Beta parameters

Large α: HDI (Highest Density Interval) more to the right
Large β: HDI rather left, lower values
Both large: (*more samples*) narrow HDI

```
install.packages("berryFunctions")
```

```
library(berryFunctions) # >= version 1.9.6 (2015-12-18)
betaPlot(shape1=alpha, shape2=beta, ...)
betaPlotComp()
```

Beta parameters

```
betaPlot(14, 2.4, ylim=lim0(5))
```

Beta parameters

```
betaPlot(6, 16, ylim=lim0(5))
```

Beta parameters

```
betaPlot(10, 10, ylim=lim0(5))
```

Beta parameters

```
betaPlot(3, 3, ylim=lim0(5))
```

Beta parameters

```
betaPlot(1, 1, ylim=lim0(5))
```

Beta parameters

```
betaPlot(0.8, 2, ylim=lim0(5))
```

Beta parameters

```
betaPlot(0.4, 0.4, ylim=lim0(5))
```

a = 0.5
a = 1
a = 2
a = 3
a = 4
a = 10
b = 0.5
b = 1
b = 2
b = 3
b = 4
b = 10
a=0.5, b=0.5
a=1, b=0.5
a=2, b=0.5
a=3, b=0.5
a=4, b=0.5
a=10, b=0.5
a=0.5, b=1
a=1, b=1
a=2, b=1
a=3, b=1
a=4, b=1
a=10, b=1
a=0.5, b=2
a=1, b=2
a=2, b=2
a=3, b=2
a=4, b=2
a=10, b=2
a=0.5, b=3
a=1, b=3
a=2, b=3
a=3, b=3
a=4, b=3
a=10, b=3
a=0.5, b=4
a=1, b=4
a=2, b=4
a=3, b=4
a=4, b=4
a=10, b=4
a=0.5, b=10
a=1, b=10
a=2, b=10
a=3, b=10
a=4, b=10
a=10, b=10
0
1
2
3
4
0.0
0.5
1.0

Beta distribution properties

Beta distribution properties

$$mean = \mu = \frac{a}{a+b}$$

Beta distribution properties

$$mean = \mu = \frac{a}{a+b} \qquad variance = \sigma^2 = \frac{ab}{(a+b)^2 * (a+b+1)}$$

Beta distribution properties

$$mean = \mu = \frac{a}{a+b} \qquad variance = \sigma^2 = \frac{ab}{(a+b)^2 * (a+b+1)}$$

With these formulas, if a and b $\gg 1$, you can estimate an approximation to the normal distribution.
To get a & b from mean & var, see
https://stats.stackexchange.com/a/12239

Beta distribution properties

$$mean = \mu = \frac{a}{a+b} \qquad variance = \sigma^2 = \frac{ab}{(a+b)^2 * (a+b+1)}$$

With these formulas, if a and b $\gg 1$, you can estimate an approximation to the normal distribution.
To get a & b from mean & var, see
https://stats.stackexchange.com/a/12239

$$mode = density\ peak = m = \frac{a-1}{a+b-2} \qquad for\ a, b > 1$$

Beta distribution properties

$$mean = \mu = \frac{a}{a+b} \qquad variance = \sigma^2 = \frac{ab}{(a+b)^2 * (a+b+1)}$$

With these formulas, if a and b $\gg 1$, you can estimate an approximation to the normal distribution.
To get a & b from mean & var, see
https://stats.stackexchange.com/a/12239

$$mode = density\ peak = m = \frac{a-1}{a+b-2} \qquad for\ a, b > 1$$

If you want to set the mode at a certain value, you can compute beta with

$$b = \frac{a-1}{m} - a + 2 \qquad \text{and alpha with} \qquad a = \frac{(2-b-1/m)*m}{m-1}$$

Beta distribution specification

Exercise 48: Beta distribution parameter determination

You want to specify a distribution with the peak at 90%, reached in a sample of 100.

1. What are the parameters?
2. What does the distribution look like? (Graph it)
3. BONUS: How does the HDI (highest density interval) width change for a sample size of 10?

Beta distribution parametrization solution

```
v <- seq(0,1, len=200)
plot(v, dbeta(v, 91,11), type="l", las=1)
lines(v, dbeta(v, 10,2), col="red")
points(qbeta(c(0.1, 0.9), 91,11), rep(-0.5,2), pch=3)
points(qbeta(c(0.1, 0.9), 10,2), rep(-0.5,2), pch=3, col="red")
```

Bayesian inference with beta distribution

Prior: weak knowledge that true proportion of cats among pets must be around 10 to 30%, but may also be more.

Bayesian inference with beta distribution

Prior: weak knowledge that true proportion of cats among pets must be around 10 to 30%, but may also be more.
Data: observation in survey: 35 of 100 households with a pet have a cat.

Bayesian inference with beta distribution

Prior: weak knowledge that true proportion of cats among pets must be around 10 to 30%, but may also be more.
Data: observation in survey: 35 of 100 households with a pet have a cat.
Posterior beta parameters: a_prior + successes, b_prior + failures

Bayesian inference with beta distribution

Prior: weak knowledge that true proportion of cats among pets must be around 10 to 30%, but may also be more.
Data: observation in survey: 35 of 100 households with a pet have a cat.
Posterior beta parameters: a_prior + successes, b_prior + failures

```
betaPlot(3, 9, ylim=lim0(10), cum=F, mar=c(2,2,0,0), keeppar=T)
lines(x, dbeta(x, 3+35, 9+65), lwd=2, col="green4")
```

Shift + scale values

If values do not lie between 0 and 1, but between m and n, you can transform the raw values (y) to the 0:1 space (y') with

$$y' = \frac{(y-m)}{(n-m)}$$

and beta values (y') to raw values (y) with

$$y = y' * (n - m) + m$$

Outlook: used for beta regression, see Liu and Kong (2015): zoib: An R Package for Bayesian Inference for Beta Regression and Zero/One Inflated Beta Regression. In: the R Journal Vol. 7/2, December 2015

Shift + scale values

```
x <- 400:580; m <- 120 ; n <- 570
plot(x, dbeta((x-m)/(n-m),30,4), type="l")
lines(x, dnorm((x-m)/(n-m),30/34, sd=sqrt(120/(34^2*35))),
      col="red")
```

Shift + scale values

```
m <- 120 ; n <- 570; set.seed(1)
x <- rbeta(800,30,4) * (n-m)+m
library(extremeStat) # github.com/brry/extremeStat
plotLfit(distLfit(x, quiet=T), breaks=30, nbest=11, legargs=c(x="left"))
```

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
   1. Exploratory Data Analysis (EDA)
   2. Normal distribution
   3. Other distribution functions
   4. Beta distribution
   5. Extreme Value Statistics
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Extreme value statistics (EVS)

More extensive script version of these slides (in German):
https://github.com/brry/rclick/blob/master/15-Extremwertstatistik.r

1. Data
2. Return period, plotting positions
3. Fitting extreme value distribution functions
4. Plotting and computing quantiles from evd
5. L-moments
6. R package `extremeStat`

EVS: Dataset

Annual streamflow maxima 1976-2010 at a stream in Austria

```
JM <- c(61.5, 77.0, 37.0, 69.3, 75.6, 74.9, 43.7, 50.8,
55.6, 84.1, 43.6, 81.9, 60.1, 72.4, 61.6, 94.8, 82.6, 57.2,
63.1, 73.8, 51.3, 93.6, 56.9, 52.1, 40.4, 48.9, 113.6,
35.4, 40.1, 89.6, 47.8, 57.6, 38.9, 69.7, 110.8)
```

EVS: Dataset

Annual streamflow maxima 1976-2010 at a stream in Austria

```
JM <- c(61.5, 77.0, 37.0, 69.3, 75.6, 74.9, 43.7, 50.8,
55.6, 84.1, 43.6, 81.9, 60.1, 72.4, 61.6, 94.8, 82.6, 57.2,
63.1, 73.8, 51.3, 93.6, 56.9, 52.1, 40.4, 48.9, 113.6,
35.4, 40.1, 89.6, 47.8, 57.6, 38.9, 69.7, 110.8)
```

Annual maxima must be independent from each other
-> no extreme event at year-separation used for both years

EVS: Dataset

Annual streamflow maxima 1976-2010 at a stream in Austria

```
JM <- c(61.5, 77.0, 37.0, 69.3, 75.6, 74.9, 43.7, 50.8,
55.6, 84.1, 43.6, 81.9, 60.1, 72.4, 61.6, 94.8, 82.6, 57.2,
63.1, 73.8, 51.3, 93.6, 56.9, 52.1, 40.4, 48.9, 113.6,
35.4, 40.1, 89.6, 47.8, 57.6, 38.9, 69.7, 110.8)
```

Annual maxima must be independent from each other
-> no extreme event at year-separation used for both years

The alternative to this Block Maxima (BM) approach is using Peak-Over-Threshold (POT) extremes.

EVS: Dataset time series graph

```
plot(1976:2010, JM, type="l", las=1, ylab="", xlab="",
     main="Annual streamflow maxima [m\U00B3/s]")
# Moving Average (in berryFunctions):
lines(1976:2010, movAv(JM), col="red", lwd=3)
# can indicate trends or cycles, but:
# window width = "researcher degree of freedom"
```

EVS: Dataset histgram

```
hist(JM, col="salmon", las=1)
```

EVS: empirical quantiles

```r
quantile(JM)

##     0%    25%    50%    75%   100%
##  35.40  49.85  61.50  76.30 113.60
```

EVS: empirical quantiles

```
quantile(JM)

##     0%    25%    50%    75%   100%
##  35.40  49.85  61.50  76.30 113.60
```

75% of the annual maxima are smaller than 76.3 m^3/s (upper quartile). This discharge is exceeded in 25 of 100 years. Statistically, we'd expect an event of this magnitude (or larger) once every

EVS: empirical quantiles

```
quantile(JM)

##     0%    25%    50%    75%   100%
##  35.40  49.85  61.50  76.30 113.60
```

75% of the annual maxima are smaller than 76.3 m^3/s (upper quartile). This discharge is exceeded in 25 of 100 years. Statistically, we'd expect an event of this magnitude (or larger) once every 4 years. Each year, there's a statistical probability of 25% for 76 m^3/s to be reached/exceeded.

EVS: empirical quantiles

```
quantile(JM)

##     0%    25%    50%    75%   100%
##  35.40  49.85  61.50  76.30 113.60
```

75% of the annual maxima are smaller than 76.3 m^3/s (upper quartile). This discharge is exceeded in 25 of 100 years. Statistically, we'd expect an event of this magnitude (or larger) once every 4 years. Each year, there's a statistical probability of 25% for 76 m^3/s to be reached/exceeded.

This all assumes a stationary runoff regime in the catchment, i.e. no relevant changes over time in rainfall patterns or landscape composition.

EVS: empirical quantiles

```
quantile(JM)

##     0%    25%    50%    75%   100%
##  35.40  49.85  61.50  76.30 113.60
```

75% of the annual maxima are smaller than 76.3 m^3/s (upper quartile). This discharge is exceeded in 25 of 100 years. Statistically, we'd expect an event of this magnitude (or larger) once every 4 years. Each year, there's a statistical probability of 25% for 76 m^3/s to be reached/exceeded.

This all assumes a stationary runoff regime in the catchment, i.e. no relevant changes over time in rainfall patterns or landscape composition.

This is also purely empirical, based on past observations only, with no assumptions on how the extrema are generally (supposed to be) distributed.

EVS: Plotting Positions + Return Period

Plotting Positions (PP) were used back in the olden days to graphically fit extreme value distributions on a Gumbel-scaled axis.

EVS: Plotting Positions + Return Period

Plotting Positions (PP) were used back in the olden days to graphically fit extreme value distributions on a Gumbel-scaled axis.
Several methods to compute PP are compared in the mentioned Rclick script.

EVS: Plotting Positions + Return Period

Plotting Positions (PP) were used back in the olden days to graphically fit extreme value distributions on a Gumbel-scaled axis.
Several methods to compute PP are compared in the mentioned Rclick script. We'll only use the method from Weibull (1939) with $Pne = \frac{R}{n+1}$, but there are also e.g. Hazen (1914) $\frac{R-0.50}{n}$, Gringorton (1963) $\frac{R-0.44}{n+0.12}$ or Beard (1943) $\frac{R-0.31}{n+0.38}$.

EVS: Plotting Positions + Return Period

Plotting Positions (PP) were used back in the olden days to graphically fit extreme value distributions on a Gumbel-scaled axis.
Several methods to compute PP are compared in the mentioned Rclick script. We'll only use the method from Weibull (1939) with $Pne = \frac{R}{n+1}$, but there are also e.g. Hazen (1914) $\frac{R-0.50}{n}$, Gringorton (1963) $\frac{R-0.44}{n+0.12}$ or Beard (1943) $\frac{R-0.31}{n+0.38}$. These all overestimate the return period and hence underestimate flood risk (compared to Weibull). E.g. for 35 years of data, Hazen assigns an RP of 70 years to the highest measured value, Weibull 36 years.

EVS: Plotting Positions + Return Period

Plotting Positions (PP) were used back in the olden days to graphically fit extreme value distributions on a Gumbel-scaled axis.
Several methods to compute PP are compared in the mentioned Rclick script. We'll only use the method from Weibull (1939) with $Pne = \frac{R}{n+1}$, but there are also e.g. Hazen (1914) $\frac{R-0.50}{n}$, Gringorton (1963) $\frac{R-0.44}{n+0.12}$ or Beard (1943) $\frac{R-0.31}{n+0.38}$. These all overestimate the return period and hence underestimate flood risk (compared to Weibull). E.g. for 35 years of data, Hazen assigns an RP of 70 years to the highest measured value, Weibull 36 years.

```
m <- sort(JM) # ascendingly sorted maxima
n <- length(m)
Rank <- 1:n
# empirical non-exceedance probability = Percentile:
Pne <- Rank / (n+1)
# RP: Return Period = annuality = 1/Pe = 1/(1-Pne)
RP <- 1/(1-Pne)
```

EVS: Plotting Positions - Graph

```
plot(m,Pne, xlab="Value", ylab="Probability of not being exceeded")
plot(ecdf(JM), ylab="empirical cumulated density")
plot(m,RP, xlab="Streamflow value", ylab="Return Period [years]")
```

EVS: Plotting Positions - Graph

```
plot(m,Pne, xlab="Value", ylab="Probability of not being exceeded")
plot(ecdf(JM), ylab="empirical cumulated density")
plot(m,RP, xlab="Streamflow value", ylab="Return Period [years]")
```

Plot 1 and 2 are almost equal in nature, but the ECDF goes from 0 to 1 exactly. It leaves no room for potentially even higher or lower values.

EVS: Plotting Positions - Graph

```
plot(m,Pne, xlab="Value", ylab="Probability of not being exceeded")
plot(ecdf(JM), ylab="empirical cumulated density")
plot(m,RP, xlab="Streamflow value", ylab="Return Period [years]")
```

Plot 1 and 2 are almost equal in nature, but the ECDF goes from 0 to 1 exactly. It leaves no room for potentially even higher or lower values.
The second-highest value is reached or exceeded twice in the 35 years. Statistically, this occurs once in every 18 years. The return period of the third-largest value is $36/3 = 12$ years. This reciprocal relationship (1/x) causes the counterintuitive leaps in RP.

EVS: Fitting the Gamma distribution

The plotting positions are irrelevant for fitting distributions. E.g. the parameters of the Gamma-distribution can be determined from the statistical moments of the observations - mean, var, skew (non-symmetry), kurtosis (non-normality).

EVS: Fitting the Gamma distribution

The plotting positions are irrelevant for fitting distributions. E.g. the parameters of the Gamma-distribution can be determined from the statistical moments of the observations - mean, var, skew (non-symmetry), kurtosis (non-normality).

```
shape <- mean(m)^2/var(m)   ;   rate  <- mean(m)/var(m)
hist(m, freq=FALSE, col="salmon",
     main="Gamma is OK for this dataset, but underestimates tail")
Q <- seq(par("usr")[1], par("usr")[2], len=200)
lines(Q, dgamma(x=Q, shape=shape, rate=rate), lwd=4)
```

EVS: Discharge over Return Periods I

```
# remember: RP   =   1/(1-Pne)   =   1/(1- Rank/(n+1) )
plot(RP, m, ylab="Discharge HQ  [m\U00B3/s]", xlab="Return Period RP  [a]",
     las=1, main="yearly streamflow extrema / return period", pch=20)
RPgamma <- 1/( 1-  pgamma( Q, shape=shape, rate=rate)    )
lines(RPgamma, Q, col="orange", lwd=2)
text(15,79, paste("mean =",round(mean(m),3),"  sd =",round(sd(m),3)),adj=0)
text(15,66, bquote("shape = "* mu^2/sigma*" = "*.(round(shape, 2))), adj=0)
text(15,53, bquote("rate = " * mu/sigma * " = "*.(round(rate,  4))), adj=0)
text(15,40, bquote("scale = "* 1/rate *   " = "*.(round(1/rate,3))), adj=0)
```

EVS: Return levels

$RP = \frac{1}{Pe} = \frac{1}{1-Pne}$

$Pne = 1 - \frac{1}{RP}$

$RL = quantile(x, \quad prob = Pne)$

```
ReturnYears <- c(5,10,20,50)
qgamma(p=1-(1/ReturnYears), shape=shape, rate=rate)

## [1]  81.043  91.927 101.597 113.233
```

EVS: Practicing with the Gumbel distribution

Exercise 49: Extreme value statistics

1. Plot a density histogram of the built-in `Nile` dataset.
2. Fit the Gamma distribution by estimating its parameters from the sample moments. $shape = \mu^2/var \quad ; \quad rate = \mu/var$
3. Fit the Gumbel distribution as well, using $scale = \sqrt{6} * \sigma/\pi \quad ; \quad loc = \mu - 0.5772 * scale$
4. Add both distributions to the graph. You can use `evd::dgumbel`.
5. BONUS: add a legend or label the lines directly
6. Plot streamflow over return period using Weibull plotting positions:
   `N_s <- sort(Nile) ; n <- length(Nile) ; RP <- 1/(1-1:n/(n+1))`
7. Using the help of `?plot.default`, use a logarithmic axis for the RP.
8. BONUS: suppress axis drawing and add `berryFunctions::logAxis(1)`
9. Add both fitted lines (in the same colors used in the previous plot)
10. From both distributions, estimate the discharge expected with RPs:
    `ReturnYears <- c(5,10,20,50,100,200)`

EVS: Solutions to EVD exercise - prob density

```
hist(Nile, freq=FALSE, col="burlywood", main="Nile", breaks=20)
shape <- mean(Nile)^2/var(Nile)   ;   rate  <- mean(Nile)/var(Nile)
scale <- sqrt(6)*sd(Nile)/pi  ;  loc <- mean(Nile)-0.5772*scale
Q <- seq(400, 1600, len=200)
lines(Q, dgamma(x=Q, shape=shape, rate=rate), lwd=4, col="blue")
lines(Q, evd::dgumbel(x=Q, loc=loc, scale=scale), lwd=4, col="red")
legend("topright", c("Gamma", "Gumbel"), col=c("blue","red"), lwd=3)
```

EVS: Solutions to EVD exercise - EV plot

```
N_s <- sort(Nile) ; n <- length(Nile)  ;  RP <- 1/(1-1:n/(n+1))
plot(RP, N_s, ylab="Nile flow", xlab="", log="x", ylim=c(500,1500),
     xlim=c(1,200), xaxt="n") ; berryFunctions::logAxis(1)
title(xlab="Return period", line=1.5)
lines(1/(1-    pgamma(q=Q, shape=shape, rate=rate)), Q, lwd=4,col="blue")
lines(1/(1-evd::pgumbel(q=Q, loc=loc, scale=scale)), Q, lwd=4, col="red")
legend("bottomright", c("Gamma", "Gumbel"), col=c("blue","red"), lwd=3)
```

EVS: Solutions to EVD exercise - return levels

```
ReturnYears <- c(5,10,20,50,100,200)
      qgamma(p=1-(1/ReturnYears), shape=shape, rate=rate)

## [1] 1058 1142 1214 1299 1358 1413

evd::qgumbel(p=1-(1/ReturnYears), loc=loc, scale=scale)

## [1] 1041 1140 1235 1358 1450 1542
```

EVS: L moments I, http://en.wikipedia.org/wiki/L-moment

Block maxima should approach the General Extreme Value Distribution (GEV)
Peaks over a threshold should approach the General Pareto Distribution (GPD).

```
# SAMLMU: Unbiased SAMple L-Moments
moments <- lmom::samlmu(JM, sort.data=TRUE, nmom=3)
# PEL: Parameter Estimation from L-moments
param <- lmom::pelgev(moments)
moments;   param

##        l_1        l_2        t_3
## 64.7800000 11.6351261  0.1379388
##          xi       alpha           k
## 55.48679492 17.54929970  0.05040469
```

EVS: L moments II

```
hist(JM, freq=FALSE, col="salmon", main="GEV")
lines(Q, evd::dgev(x=Q, loc=param["xi"],
                   scale=param["alpha"], shape=param["k"]),
      col="purple", lwd=3)
```

```
lmom::quagev(f=0.99, param) # 99% of values under 127.5
```

EVS: extremeStat

Read the extremeStat vignette, especially the last section for more info.

```
library(extremeStat)
Q50 <- distLextreme(JM, gpd=FALSE, quiet=TRUE)
quantile(Q50$returnlev[1:16,"RP.50"])

##       0%      25%      50%      75%     100%
## 107.1339 113.9177 115.4346 117.5186 132.5435
```

107-132, quite a range! Most suitable distributions agree around 115.

EVS: extremeStat plots: streamflow - return period

```
plotLextreme(Q50)
```

EVS: extremeStat plots: streamflow distribution

```
plotLfit(Q50, nbest=17, legargs=list(x="topright"))
```

EVS: extremeStat plots: streamflow CDF

```
plotLfit(Q50, cdf=TRUE, legargs=list(x="bottomright"))
```

EVS: use extremeStat

Exercise 50: extremeStat

1. Using distLextreme, printL and plotLextreme in extremeStat, explore the extremes of the `Nile` streamflow.
2. What streamflow is expected once per 200 years?
3. BONUS 1: get uncertainty ranges for the GEV estimate with `distLexBoot`
4. BONUS 2: use the POT method with a useful truncation

EVS: use extremeStat

```
nile <- distLextreme(Nile, RPs=c(50,100,200), quiet=TRUE)
head(nile$returnlev)

##     RP.50 RP.100 RP.200
## wak  1287   1316   1337
## ray  1322   1384   1442
## kap  1301   1343   1377
## gum  1379   1476   1572
## wei  1315   1374   1429
## gev  1327   1394   1455
```

```
printL(nile)
plotLextreme(nile)
plotLweights(nile)
nile_b <- distLexBoot(nile, selection="gev")
```

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
   1. Simpsons Paradox
   2. Normality tests
   3. Confidence Intervals
   4. (Linear) regression
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Simpson's Paradox

- Kievit et al (2015): Simpson's paradox in psychological science: a practical guide (Frontiers in Psychology)
- vudlab app
- Wikipedia entry
- 2016 example: development of crime rates in the USA
- minutephysics video (2017)

Simpson's Paradox: Kidney stone treatment success rate

	open surgery	small puncture
small stones large stones		
together	$\frac{273}{350} = 78\%$	$\frac{289}{350} = \mathbf{83\%}$

Wikipedia, Julious + Mullee (1994)

Simpson's Paradox: Kidney stone treatment success rate

	open surgery	small puncture
small stones	$\frac{81}{87} = \mathbf{93\%}$	$\frac{234}{270} = 87\%$
large stones		
together	$\frac{273}{350} = 78\%$	$\frac{289}{350} = \mathbf{83\%}$

Wikipedia, Julious + Mullee (1994)

Simpson's Paradox: Kidney stone treatment success rate

	open surgery	small puncture
small stones	$\frac{81}{87} = \mathbf{93\%}$	$\frac{234}{270} = 87\%$
large stones	$\frac{192}{263} = \mathbf{73\%}$	$\frac{55}{80} = 69\%$
together	$\frac{273}{350} = 78\%$	$\frac{289}{350} = \mathbf{83\%}$

Wikipedia, Julious + Mullee (1994)

Simpson's Paradox: Kidney stone treatment success rate

	open surgery	small puncture
small stones	$\frac{81}{87} = \mathbf{93\%}$	$\frac{234}{270} = 87\%$
large stones	$\frac{192}{263} = \mathbf{73\%}$	$\frac{55}{80} = 69\%$
together	$\frac{273}{350} = 78\%$	$\frac{289}{350} = \mathbf{83\%}$

Wikipedia, Julious + Mullee (1994)

In less severe cases (small kidney stones), doctors favor percutaneous nephrolithotomy (which involves only a small puncture). Doctors tend to give the severe cases (large stones) the better treatment (open surgery). The success rate is more strongly influenced by the severity of the case than by the choice of treatment. When the less effective treatment (small puncture) is applied more frequently to less severe cases, it can appear to be a more effective treatment.

Simpson's Exercise

Exercise 51: Simpson

Visualize farming.txt *(rightclick **Raw**, save as)*, a fictional dataset measuring crop yield on 30 study fields after pesticide application.

Farming solution I

Farming solution II

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
   1. Simpsons Paradox
   2. Normality tests
   3. Confidence Intervals
   4. (Linear) regression
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Normality Test: Kolmogorov-Smirnov

Normality Test: Kolmogorov-Smirnov

- If a population is normally distributed, it is described by only two parameters: the mean (position) and the sd (width, dispersion) of the bell shaped curve.

Normality Test: Kolmogorov-Smirnov

- If a population is normally distributed, it is described by only two parameters: the mean (position) and the sd (width, dispersion) of the bell shaped curve.
- This is an important assumption for many classical statistical methods.

Normality Test: Kolmogorov-Smirnov

- If a population is normally distributed, it is described by only two parameters: the mean (position) and the sd (width, dispersion) of the bell shaped curve.
- This is an important assumption for many classical statistical methods.
- Whether a dataset is normally distributed can be checked with a histogram (visually effective, but the class limits are subjective), with qqplots (I don't find them very intuitive), or with statistical tests.

Normality Test: Kolmogorov-Smirnov

- If a population is normally distributed, it is described by only two parameters: the mean (position) and the sd (width, dispersion) of the bell shaped curve.
- This is an important assumption for many classical statistical methods.
- Whether a dataset is normally distributed can be checked with a histogram (visually effective, but the class limits are subjective), with qqplots (I don't find them very intuitive), or with statistical tests.
-

```
data <- rnorm(1000, mean=97, sd=8.9)
shapiro.test(data)
ks.test(data, "pnorm", mean(data), sd(data))
# if p > 0.05: accept Nullhypothesis that data are normally distributed.

if(!requireNamespace("nortest")) install.packages("nortest")
library(nortest)          # with Lilliefors-correction!
help(package="nortest")
lillie.test(data)  # Lilliefors (Kolmogorov-Smirnov) test for normality
ad.test(data)      # Anderson-Darling test for normality
cvm.test(data)     # Cramer-von Mises test for normality
pearson.test(data) # Pearson chi-square test for normality
sf.test(data)      # Shapiro-Francia test for normality
```

Normality Test: Kolmogorov-Smirnov

KS Hypothesis Test: Reject the null hypothesis (H0) that the values are drawn from a normal distribution if $D < \frac{k_\alpha}{\sqrt{n}}$.

Normality Test: Kolmogorov-Smirnov

KS Hypothesis Test: Reject the null hypothesis (H0) that the values are drawn from a normal distribution if $D < \frac{k_\alpha}{\sqrt{n}}$.

$D = max(\mid F_{(n)}(x) - F(x) \mid)$ *Difference between empirical and statistical (theoretical, population) Cumulated Density Function (CDF)*

Normality Test: Kolmogorov-Smirnov

KS Hypothesis Test: Reject the null hypothesis (H0) that the values are drawn from a normal distribution if $D < \frac{k_\alpha}{\sqrt{n}}$.

$D = max(\mid F_{(n)}(x) - F(x) \mid)$ *Difference between empirical and statistical (theoretical, population) Cumulated Density Function (CDF)*

$k_\alpha = \frac{\sqrt{-0.5 * ln(\frac{\alpha}{2})}}{\sqrt{n}}$ $(for\ n > 35)$ α often 0.05 (95% confidence)

Normality Test: Kolmogorov-Smirnov

KS Hypothesis Test: Reject the null hypothesis (H0) that the values are drawn from a normal distribution if $D < \frac{k_\alpha}{\sqrt{n}}$.

$D = max(\mid F_{(n)}(x) - F(x) \mid)$ *Difference between empirical and statistical (theoretical, population) Cumulated Density Function (CDF)*

$k_\alpha = \frac{\sqrt{-0.5 * ln(\frac{\alpha}{2})}}{\sqrt{n}}$ $(for\ n > 35)$ α often 0.05 (95% confidence)

Normality Test: Kolmogorov-Smirnov

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
   1. Simpsons Paradox
   2. Normality tests
   3. Confidence Intervals
   4. (Linear) regression
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Background confidence intervals

Background confidence intervals

- About 70% of the values in a Gaussian distribution lie between mean-1sd and mean+1sd, as is shown by `pnorm(1) - pnorm(-1)`: `0.6826895`.

Background confidence intervals

- About 70% of the values in a Gaussian distribution lie between mean-1sd and mean+1sd, as is shown by `pnorm(1) - pnorm(-1)`: `0.6826895`.
- 95% of the values are between mean $\pm$ 1.96 sd, as is shown by `qnorm(0.975)`: `1.959964`. (As the bell curve is symmetrical, we want to know below what value 97.5% of the values are in the cumulated density).

Background confidence intervals

- About 70% of the values in a Gaussian distribution lie between mean-1sd and mean+1sd, as is shown by `pnorm(1) - pnorm(-1)`: `0.6826895`.
- 95% of the values are between mean $\pm$ 1.96 sd, as is shown by `qnorm(0.975)`: `1.959964`. (As the bell curve is symmetrical, we want to know below what value 97.5% of the values are in the cumulated density).
- This is the famous parameter in calculating the limits of confidence intervals (CI).

$$CI = \mu \pm p * \frac{sd}{\sqrt{n}}$$

Background confidence intervals

- About 70% of the values in a Gaussian distribution lie between mean-1sd and mean+1sd, as is shown by `pnorm(1) - pnorm(-1)`: `0.6826895`.
- 95% of the values are between mean $\pm$ 1.96 sd, as is shown by `qnorm(0.975)`: `1.959964`. (As the bell curve is symmetrical, we want to know below what value 97.5% of the values are in the cumulated density).
- This is the famous parameter in calculating the limits of confidence intervals (CI).

$$CI = \mu \pm p * \frac{sd}{\sqrt{n}}$$

$$p = \texttt{qt(1-alpha/2, df=n-1)}$$

Parameter confidence intervals

en.wikipedia.org/wiki/Student's_t-distribution#Table_of_selected_values
For a two-sided confidence interval with `qt(1-alpha/2, df=n-1)`:

	a=0.01	a=0.05	a=0.10
n=10	3.2498	2.2622	1.8331
n=30	2.7564	2.0452	1.6991
n=50	2.6800	2.0096	1.6766
n=100	2.6264	1.9842	1.6604
n=1000	2.5808	1.9623	1.6464

Parameter confidence intervals

en.wikipedia.org/wiki/Student's_t-distribution#Table_of_selected_values
For a two-sided confidence interval with `qt(1-alpha/2, df=n-1)`:

	a=0.01	a=0.05	a=0.10
n=10	3.2498	2.2622	1.8331
n=30	2.7564	2.0452	1.6991
n=50	2.6800	2.0096	1.6766
n=100	2.6264	1.9842	1.6604
n=1000	2.5808	1.9623	1.6464

code for generating this table:

```
alpha <- c(0.01, 0.05, 0.10)
n <- c(10, 30, 50, 100, 1000)
ttab <- t(sapply(n, function(x) qt(1-alpha/2 ,df=x-1)))
colnames(ttab) <- paste0("a=",alpha)
rownames(ttab) <- paste0("n=",n)
ttab
```

CI example

Exercise 52: Confidence intervals

1. Comment the code below.
2. plot the points along a number line and add the mean value with `points` and the confidence interval with `arrows` with `angle=90, code=3`. BONUS: plot the points as filled dots in a semitransparent color, see `rgb`.

```
values <- rnorm(80, 53, 4)
m <- mean(values)   ; m #
v <- var(values)    ; v #
n <- length(values) ; n #
p <- qnorm(0.975)   ; p #
p <- qt(0.975, df=n-1)  ; p #
cl <- m - p * sqrt(v/n) ; cl #
cu <- m + p * sqrt(v/n) ; cu #
```

CI in a faster way

Scripting this manually needs several LOC (lines of code):

```
m <- mean(values)
v <- var(values)
n <- length(values)
p <- qt(0.975, df=n-1)

m + c(-1,1)* p * sqrt(v/n)

## [1] 52.34151 54.00713
```

CI in a faster way

Scripting this manually needs several LOC (lines of code):

```
m <- mean(values)
v <- var(values)
n <- length(values)
p <- qt(0.975, df=n-1)

m + c(-1,1)* p * sqrt(v/n)

## [1] 52.34151 54.00713
```

You can use the side effect of `t.test`:

```
t.test(values, conf.level=.95)$conf.int

## [1] 52.34151 54.00713
## attr(,"conf.level")
## [1] 0.95
```

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
   1. Simpsons Paradox
   2. Normality tests
   3. Confidence Intervals
   4. (Linear) regression
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Linear Regression

This section (linear models) was originally written by Matthias Seibert (GFZ).
It is excluded from this main document and can be obtained as pdf.

Storks Deliver Babies ($p = 0.008$)

p-hacking

Fit multiple functions: berryFunctions::mReg. See the mReg sections in the Vignette and Rclick page

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
    1. Date and time formatting
    2. Time series
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Char to date / time and back

Exercise 53: char time

1. Convert the character string "2017-11-29 13:41:16" to a time object with `?strptime` (string parsed to time). Check the `str`ucture.
2. BONUS: reflect the local or GMT time zone.
3. `?paste` the following charstrings together "14.2.2016" and "17:18", separated with a minus symbol.
4. Convert the result with `strptime`. Convert it to a number (what does it mean?) and with `as.POSIXct`.
5. BONUS: format the current `Sys.Date()` as "Mi, 29.Nov.2017 (= ein Mittwoch im November)". On what weekday were you born?
6. BONUS: Print the names in English, even if the locale is German. See Rclick chapter 13 on `?Sys.setlocale`.

Char - Time Solutions

```
now <- strptime("2017-11-29 13:41:16", format="%Y-%m-%d %H:%M:%S")
now ; class(now); str(now)

## [1] "2017-11-29 13:41:16 CET"
## [1] "POSIXlt" "POSIXt"
##  POSIXlt[1:1], format: "2017-11-29 13:41:16"

strptime("2017-11-29 13:41:16", format="%F %T", tz="UTC")

## [1] "2017-11-29 13:41:16 UTC"
```

Char - Time Solutions

```
now <- paste("14.2.2016", "17:19", sep="-")
now <- strptime(now, "%d.%m.%Y-%H:%M")
as.numeric(now) # number of seconds since 1970-01-01

## [1] 1455466740

str(as.POSIXct(now))

##  POSIXct[1:1], format: "2016-02-14 17:19:00"

format( Sys.Date(), "%a, %d.%b.%Y (= some %A in %B)")

## [1] "Tue, 26.May.2020 (= some Tuesday in May)"

format(as.Date("1997-04-13"), "%A") # Born on Sunday

## [1] "Sunday"
```

Time sequences

Exercise 54: equally spaced time sequences

1. Create a weekly sequence with `seq()` starting
```
t1 <- as.Date("2007/10/18") # and ending at
t2 <- as.Date("2007/11/25")
```
2. BONUS: How about monthly sequences? (by = number of days not helpful)
3. From the time range "tr", create a sequence in steps of 15 minutes:
```
tr <- strptime(c("05.08.2010, 12:15",
                 "09.02.2013, 14:45"),
               format="%d.%m.%Y, %H:%M")
```
4. Create a data.frame with that time sequence and values from a random walk (see `rnorm` and `cumsum`). Both columns should be named aptly.
5. BONUS: Correct the by argument in `seq(as.Date("2017-08-30"), as.Date("2017-06-30"), by=7)`.

Time sequence solutions

```
seq(t1, t2, by=7)

## [1] "2007-10-18" "2007-10-25" "2007-11-01"
## [4] "2007-11-08" "2007-11-15" "2007-11-22"

seq(t1,t2+150, by="month") # uses seq.Date

## [1] "2007-10-18" "2007-11-18" "2007-12-18"
## [4] "2008-01-18" "2008-02-18" "2008-03-18"
## [7] "2008-04-18"

tseq <- seq(tr[1], tr[2], by="15 min")
fakeTS <- data.frame(time=tseq,
                     values=cumsum(rnorm(length(tseq)))
                     )
# str(fakeTS)
```

Time Series aggregation

Exercise 55: TS aggregation

1. Plot the random walk time series from the previous task. BONUS: label the axis nicely with `berryFunctions::monthAxis()`
2. Add a column uniquely identifying the month of each data point.
3. Compute monthly averages with `?tapply`
4. BONUS: Browse through Rclick chapter 13

TS aggregation solutions

```
plot(fakeTS, type="l", xaxt="n", las=1)
berryFunctions::monthAxis(mcex=0.5)
#
fakeTS$month <- format(fakeTS$time, "%Y-%m-15")
agg_month <- tapply(X=fakeTS$values, INDEX=fakeTS$month,
                    FUN=mean)
# compute mean of values, split into month groups

head(agg_month) # View(data.frame(agg_month))

## 2010-08-15 2010-09-15 2010-10-15 2010-11-15
##  -31.73680  -64.85758  -73.73245 -109.02650
## 2010-12-15 2011-01-15
## -141.99899 -137.62028
```

TS aggregation solutions

Task

Task

Your professor asks you to visually examine temperature and precipitation trends at certain meteorological stations in Germany. She hands you the files Potsdam.txt and Zugspitze.txt *(rightclick **Raw**, save as)*. How do you approach the project?

Task

Your professor asks you to visually examine temperature and precipitation trends at certain meteorological stations in Germany. She hands you the files Potsdam.txt and Zugspitze.txt *(rightclick **Raw**, save as)*. How do you approach the project?

Data selection, download, processing and saving:

```
library(rdwd) # github.com/brry/rdwd
links <- selectDWD(c("Potsdam","Zugspitze"), current=TRUE,
                   res="monthly", var="kl", per="hist", outvec=TRUE)
clim <- dataDWD(links)
write.table(clim[[1]], file="data/Potsdam.txt", row.names=F, quote=F)
write.table(clim[[2]], file="data/Zugspitze.txt", row.names=F, quote=F)
```

Task

Your professor asks you to visually examine temperature and precipitation trends at certain meteorological stations in Germany. She hands you the files Potsdam.txt and Zugspitze.txt *(rightclick **Raw**, save as)*. How do you approach the project?

Data selection, download, processing and saving:

```
library(rdwd) # github.com/brry/rdwd
links <- selectDWD(c("Potsdam","Zugspitze"), current=TRUE,
                   res="monthly", var="kl", per="hist", outvec=TRUE)
clim <- dataDWD(links)
write.table(clim[[1]], file="data/Potsdam.txt", row.names=F, quote=F)
write.table(clim[[2]], file="data/Zugspitze.txt", row.names=F, quote=F)
```

```
potsdam$date <- as.Date(as.character(potsdam$MESS_DATUM_BEGINN), "%Y%m%d")
```

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
    1. Date and time formatting
    2. Time series
11. Spatial data and GIS functionality
12. Final tasks
Course plan

Authorship note

This section (time series) was originally written by Matthias Seibert (GFZ). It is currently excluded from this main document but can be obtained as pdf.

```
library(xts)
```

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
    1. Geostatistics: Kriging
    2. R as GIS
    3. NETcdf files
12. Final tasks
Course plan

Kriging: Overview

Geostatistics: Kriging - spatial interpolation between points, using semivariance

1. Packages
2. read shapefile
3. Variogram
4. Kriging
5. Plotting

This material has also been featured by Packt publishing: datahub.packtpub.com/analytics/kriging-interpolation-geostatistics

Kriging: packages

```
# install packages if needed:
if(!requireNamespace("pacman")) install.packages("pacman")
pacman::p_load(rgeos, sf, geoR, berryFunctions)
```

Alternatively:

```
library(sf)    # for st_read (read shapefiles),
               # st_centroid, st_area, st_union
library(geoR) # as.geodata, variog, variofit,
               # krige.control, krige.conv, legend.krige
```

Kriging: read shapefile / few points for demonstration

```
x <- c(1,1,2,2,3,3,3,4,4,5,6,6,6)
y <- c(4,7,3,6,2,4,6,2,6,5,1,5,7)
z <- c(5,9,2,6,3,5,9,4,8,8,3,6,7)
plot(x,y, pch="+", cex=z/4)
```

Kriging: read shapefile II

```
GEODATA <- as.geodata(cbind(x,y,z))
plot(GEODATA)
```

Kriging: Variogram I

```
EMP_VARIOGRAM <- variog(GEODATA)

## variog: computing omnidirectional variogram

FIT_VARIOGRAM <- variofit(EMP_VARIOGRAM)

## variofit: covariance model used is matern
## variofit: weights used: npairs
## variofit: minimisation function used: optim

## Warning in variofit(EMP_VARIOGRAM): initial values not provided - running the
default search

## variofit: searching for best initial value ... selected values:
##                sigmasq phi    tausq kappa
## initial.value "9.19"  "3.65" "0"   "0.5"
## status        "est"   "est"  "est" "fix"
## loss value: 401.578968904954
```

Kriging: Variogram II

```
plot(EMP_VARIOGRAM)
lines(FIT_VARIOGRAM)
```

Kriging: Kriging

```
res <- 0.1
grid <- expand.grid(seq(min(x),max(x),res),
                    seq(min(y),max(y),res))
krico <- krige.control(type.krige="OK",
                       obj.model=FIT_VARIOGRAM)
krobj <- krige.conv(GEODATA,
                    locations=grid, krige=krico)

## krige.conv: model with constant mean
## krige.conv: Kriging performed using global neighbourhood

# KRigingObjekt
```

Kriging: Plotting I

```
image(krobj, col=rainbow2(100))
legend.krige(col=rainbow2(100),
             x.leg=c(6.2,6.7), y.leg=c(2,6),
             vert=T, off=-0.5,
             values=krobj$predict)
contour(krobj, add=T)
colPoints(x,y,z, col=rainbow2(100), legend=F)
points(x,y)
```

Kriging: Plotting II

Kriging: Plotting III

```r
library("berryFunctions") # scatterpoints by color
colPoints(x,y,z, add=F, cex=2, legargs=list(y1=0.8,y2=1))
```

Kriging: Plotting IV

```
colPoints(grid[ ,1], grid[ ,2], krobj$predict, add=F,
          cex=2, col2=NA, legargs=list(y1=0.8,y2=1))
```

Kriging: All in one script, for reference

```
if(!requireNamespace("pacman", quietly=TRUE)) install.packages("pacman")
pacman::p_load(geoR, berryFunctions) # installs packages if needed
# berryFunctions: seqPal, distance, seqR, colPoints
# geoR: as.geodata, variofit, variog, krige.conv, krige.control

K_X <- c(4,4,5,5,6,6,6,7,7,8,9,9,9)
K_Y <- c(6,9,5,8,4,6,8,4,8,7,3,7,9)
K_Z <- c(5,9,2,6,3,5,9,4,8,8,3,6,7)
K_COLS <- seqPal()
K_MDIST <- max(distance(K_X,K_Y))
K_RESOLUTION <- median(distance(K_X,K_Y))/10

K_GEODATA <- as.geodata(cbind(K_X,K_Y,K_Z))
K_VARIOGRAM <- variofit(variog(K_GEODATA, max.dist=K_MDIST))
K_GRID <- expand.grid(seqR(K_X, by=K_RESOLUTION, extend=0.1),
                      seqR(K_Y, by=K_RESOLUTION, extend=0.1))
K_OK <- krige.conv(K_GEODATA, locations=K_GRID,
        krige=krige.control(type.krige="OK", obj.model=K_VARIOGRAM))

geoR:::image.kriging(K_OK, col=K_COLS)
colPoints(K_X, K_Y, K_Z, col=K_COLS, zlab="Kriging", cex=1)
points(K_X, K_Y)

# optional: interactive map:
pacman::p_load(raster, mapview) # installs packages if needed
K_RASTER <- raster::rasterFromXYZ(cbind(K_GRID, temp=K_OK$predict),
                          crs="+proj=longlat +datum=WGS84 +no_defs")
mapview::mapview(K_RASTER, col.regions=seqPal(256))
```

Time for a real dataset

Precipitation from ca 250 gauges in Brandenburg as Thiessen Polygons with steep gradients at edges:

Time for a real dataset

Exercise 56: Kriging

1. Load and plot the shapefile in PrecBrandenburg.zip with `sf::st_read`.
2. With `colPoints` in the package `berryFunctions`, add the precipitation values at the centroids of the polygons.
3. Calculate the variogram and fit a semivariance curve.
4. Perform kriging on a grid with a useful resolution (keep in mind that computing time rises exponentially with grid size).
5. Plot the interpolated values with `image` or an equivalent (Rclick 4.15) and add contour lines.
6. What went wrong? (if you used the defaults, the result will be dissatisfying.) How can you fix it?

Solution for exercise 56.1-2: Kriging Data

```
# Shapefile:
p <- sf::st_read("data/PrecBrandenburg/niederschlag.shp",
                 quiet=TRUE)
# Plot prep
pcol <- colorRampPalette(c("red","yellow","blue"))(50)
clss <- berryFunctions::classify(p$P1, breaks=50)$index
# Plot
par(mar = c(1, 1, 1.2, 1)) # for sf plots / colPointsLegend
sf:::plot.sf(p, col=pcol[clss], max.plot=1) # P1: Precipitation
# kriging coordinates
cent <- sf::st_centroid(p)

## Warning in st_centroid.sf(p):  st_centroid assumes attributes
are constant over geometries of x

berryFunctions::colPoints(cent$x, cent$y, p$P1, add=T, cex=0.7,
                          legargs=list(y1=0.8,y2=1), col=pcol)
points(cent$x, cent$y, cex=0.7)
```

Solution for exercise 56.3: Variogram

```
library(geoR)
# Semivariance:
geoprec <- as.geodata(cbind(cent$x,cent$y,p$P1))
vario <- variog(geoprec, max.dist=130000)

## variog: computing omnidirectional variogram

fit <- variofit(vario)

## variofit: covariance model used is matern
## variofit: weights used: npairs
## variofit: minimisation function used: optim

## Warning in variofit(vario):  initial values not provided - running the default
search

## variofit: searching for best initial value ... selected values:
##                sigmasq   phi        tausq kappa
## initial.value "1326.72" "19999.93" "0"   "0.5"
## status        "est"     "est"      "est" "fix"
## loss value: 107266266.76371
```

Solution for exercise 56.3: Variogram

```
plot(vario) ; lines(fit)
# distance to closest other point:
d <- sapply(1:nrow(cent), function(i) min(berryFunctions::distance(
                          cent$x[i], cent$y[i], cent$x[-i], cent$y[-i])))
hist(d/1000, breaks=20, main="distance to closest gauge [km]")
mean(d) # 8 km  - grid resolution in next slide

## [1] 8165.633
```

Solution for exercise 56.4-5: Kriging

```
# Kriging:
res <- 1000 # 1 km, since stations are 8 km apart on average
grid <- expand.grid(seq(min(cent$x),max(cent$x),res),
                    seq(min(cent$y),max(cent$y),res))
krico <- krige.control(type.krige="OK", obj.model=fit)
krobj <- krige.conv(geoprec, locations=grid, krige=krico)

## krige.conv: model with constant mean
## krige.conv: Kriging performed using global neighbourhood

# Set values outside of Brandenburg to NA:
grid_sf <- sf::st_as_sf(grid, coords=1:2, crs=sf::st_crs(p))
isinp <- sapply(sf::st_within(grid_sf, p), length) > 0
krobj2 <- krobj
krobj2$predict[!isinp] <- NA
```

Solution for exercise 56.5: Kriging Visualization

```
geoR:::image.kriging(krobj2, col=pcol)

## Error in image.default(x = c(3252470, 3253470, 3254470,
3255470, 3256470, :  object 'pcol' not found

colPoints(cent$x, cent$y, p$P1, col=pcol, zlab="Prec", cex=0.7,
     legargs=list(y1=0.1,y2=0.8, x1=0.78, x2=0.87, horizontal=F))

## Error in plot.xy(xy.coords(x, y), type = type, ...):  plot.new
has not been called yet

plot(p, add=T, col=NA, border=8)#; points(cent$x,cent$y, cex=0.8)

## Error in polypath(p_bind(x[[i]]), border = border[i], lty =
lty[i], lwd = lwd[i], :  plot.new has not been called yet
```

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
    1. Geostatistics: Kriging
    2. R as GIS
    3. NETcdf files
12. Final tasks
Course plan

interactive shapefile editing - http://r-spatial.org

interactive shapefile editing - http://r-spatial.org

This is R. There is no if. Only how.

interactive shapefile editing - http://r-spatial.org

This is R. There is no if. Only how.

```
# Data and Packages:
download.file(
  "https://github.com/brry/course/raw/master/data/PrecBrandenburg.zip",
  destfile="prec.zip")
unzip("prec.zip")

library(sf)
library(mapview)
library(mapedit)

p <- sf::st_read("data/PrecBrandenburg/niederschlag.shp")
map <- mapview::mapview(p)
map

p2 <- mapedit::editMap(map) # add polygons, lines, points, rectangles
mapview::mapview(p2$finished)
names(p)[2] = "x1" # x column problem, see Issue 2017-12-05
p3 <- mapedit::editFeatures(p)
?franconia # from mapview package
ed <- mapedit::editFeatures(franconia[1:5,])
```

shapefile operations

sf replaces the old sp package since 2017 (sp will be deprecated).
Vignettes:

1. intro: simple features, geometry, reading/writing, projections, units
2. reading/writing, conversion
3. transformation, coordinate reference systems, geometrical operations
4. subsetting, aggregation, joining by attribute / geometry
5. plotting, projections
6. miscellaneous

point data (coordinates) - leaflet

```
ddd <- read.table(header=TRUE, sep=",", as.is=TRUE, text="
lat, lon, city, LOA # LOA: Level of Awesomeness
48.858271,   2.294245, Paris , 5
51.503162,  -0.131082, London, 3
52.514687,  13.350012, Berlin, 6
52.407160,  12.972761, Golm  , 9
")
ddd$display <- berryFunctions::popleaf(ddd)
```

point data (coordinates) - leaflet

```
ddd <- read.table(header=TRUE, sep=",", as.is=TRUE, text="
lat, lon, city, LOA # LOA: Level of Awesomeness
48.858271,   2.294245, Paris , 5
51.503162,  -0.131082, London, 3
52.514687,  13.350012, Berlin, 6
52.407160,  12.972761, Golm  , 9
")
ddd$display <- berryFunctions::popleaf(ddd)
```

Interactive plots:

```
library(leaflet)
leaflet(ddd) %>% addTiles() %>%
  addCircleMarkers(~lon, ~lat, popup=~display)
```

Different background maps

point data (coordinates) - OSMscale

Static maps: (OSMscale installation instructions)

```
library(OSMscale)
pointsMap(lat, lon, data=ddd, quiet=TRUE)
```

R geo, further material

Use the sf, mapview, osmar and osmplotr packages!

Combine city names with geolocations:
blog.revolutionanalytics.com/2015/11/marriott.html

For this section (spatial data / GIS), a lot was originally written by Matthias Seibert (GFZ). With the advance of the `sf` package (late 2016), that material is mostly outdated. Until new material is created here, his slides can be obtained as pdf.

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
    1. Geostatistics: Kriging
    2. R as GIS
    3. NETcdf files
12. Final tasks
Course plan

read nc files: ncdf4::nc_open

Install mhmVis with the instructions on github:
github.com/brry/mhmVis
Get the example file: tavg.zip

```
unzip("data/tavg.zip", exdir="data")
library(mhmVis)
nc <- read_nc("data/tavg.nc")

## nc file has been read.  Now extracting lat, lon, tavg.
This may take a minute.
## I'm done!  (took 0.27 secs)
```

visualize nc files: mhmVis::vis_nc

```
vis_nc(nc, cex=11.5)
```

visualize nc files

`mhmVis::vis_nc_all` and `mhmVis::vis_nc_film`

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
    1. Practice tasks
    2. Exam task

Course plan

Exercise 57: Final exercises to solve at home I

1. What is the difference between `c(5,3,-8)` and `vec <- c(5,3,-8)`?
2. What methods do you know to generate a vector?
3. What are object, element, command, argument, assignment, Workspace, Working Directory?
4. What data and object types do you know?
5. How do you call the help about the `lines` command?
6. How do you read a text file? What are the main arguments of the function?
7. How do you examine the imported table?
8. How do you obtain / change subsets of the table (column, row, element)?

Exercise 58: Final exercises to solve at home II

1. How often is Peter Pan mentioned in the E-book? peterpan.txt *(rightclick **Raw**, save as)*. Exclude the meta data about the book!
2. Interpret the results from `pairs(iris)` and `cor(iris[ ,1:4])`. Why does `cor(iris)` not work?
3. What are the most important functions and arguments for creating graphics?
4. Write a for loop including a while statement to distribute points randomly, but with a certain minimal distance to each other, along a line (segment sampling design). A solution is in my RclickHandbuch Chapter 10.4.
5. What functions and packages are useful for dealing with time series?

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks
    1. Practice tasks
    2. Exam task

Course plan

Exercise 59: Final exam

1. Download a long time series (e.g. weather data from DWD, NOAA), see slide 146.
2. Read the data into R and control the data types. Change the date (or time) to appropriate types with `as.Date`, `as.POSIXct` or `strptime`.
3. Write a subsection of the data (eg your birth year) to a tab-separated file so it's easy to copypaste this to Excel/Calc.
4. Plot the development of some aggregate (e.g. monthly 90% Quantile of windspeed, annual precipitation sum or temperature mean depending on hour of the day). Hints: `tapply`, `strftime`, `berryFunctions::monthLab`
5. Examine `cor`relations between variables.
6. Write a small report on your data analysis project with max 3 pages, including (beautiful) graphs. You can mail the code in a separate file.
   This is a requirement for the participation certificate with credit points.
7. BONUS 1: Write the report via Rnw, Rhtml, Rmd or Rpres (see slide 145).
8. BONUS 2: Make some interactive stuff (e.g. user can choose decade of analysis), e.g. via HTMLwidgets or Shiny (see more). This is a lot of initial effort, but you'll enable an awesome interactive graphical analyses!

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks

Course plan
- Session 2/4: Objects, file reading, Rstudio
- Session: if-else conditionals, for loops
- Session: functions and apply loops
- Session x: Recap all
- Feedback

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks

Course plan
- Session 2/4: Objects, file reading, Rstudio
- Session: if-else conditionals, for loops
- Session: functions and apply loops
- Session x: Recap all
- Feedback

to R or not to R

to R or not to R

Lucius Annaeus Seneca (4BC - 65AD):

to R or not to R

Lucius Annaeus Seneca (4BC - 65AD):

To err is human

to R or not to R

Lucius Annaeus Seneca (4BC - 65AD):

To err is human

to R is better

Session aim (Chapter 7.1 + 7.2)

Session aim (Chapter 7.1 + 7.2)

- conditional code execution (if-else)

Session aim (Chapter 7.1 + 7.2)

- conditional code execution (if-else)
- repetitional coding (for-loops)

Session aim (Chapter 7.1 + 7.2)

- conditional code execution (if-else)
- repetitional coding (for-loops)

afterwards, you should be able to:

Session aim (Chapter 7.1 + 7.2)

- conditional code execution (if-else)
- repetitional coding (for-loops)

afterwards, you should be able to:

- make reasonable guesses about logical values by looking at code

Session aim (Chapter 7.1 + 7.2)

- conditional code execution (if-else)
- repetitional coding (for-loops)

afterwards, you should be able to:

- make reasonable guesses about logical values by looking at code
- write conditionally executed code using if/else

Session aim (Chapter 7.1 + 7.2)

- conditional code execution (if-else)
- repetitional coding (for-loops)

afterwards, you should be able to:

- make reasonable guesses about logical values by looking at code
- write conditionally executed code using if/else
- decide whether to use `if(cond) A else B` or `ifelse(cond, A, B)`

Session aim (Chapter 7.1 + 7.2)

- conditional code execution (if-else)
- repetitional coding (for-loops)

afterwards, you should be able to:

- make reasonable guesses about logical values by looking at code
- write conditionally executed code using if/else
- decide whether to use `if(cond) A else B` or `ifelse(cond, A, B)`
- explain the concept of a loop

Session aim (Chapter 7.1 + 7.2)

- conditional code execution (if-else)
- repetitional coding (for-loops)

afterwards, you should be able to:

- make reasonable guesses about logical values by looking at code
- write conditionally executed code using if/else
- decide whether to use `if(cond) A else B` or `ifelse(cond, A, B)`
- explain the concept of a loop
- write simple for-loops

Session if+for Recap

- `if(TRUE) message("Yo!") else message("Dude...")`

Session if+for Recap

- `if(TRUE) message("Yo!") else message("Dude...")`
- `ifelse(c(T,T,F,T), 7, else 7.3)` for vectors

Session if+for Recap

- `if(TRUE) message("Yo!") else message("Dude...")`
- `ifelse(c(T,T,F,T), 7, else 7.3)` for vectors
- for loops repeat a block of code with a different `i` each time

Session if+for Recap

- `if(TRUE) message("Yo!") else message("Dude...")`
- `ifelse(c(T,T,F,T), 7, else 7.3)` for vectors
- for loops repeat a block of code with a different `i` each time
- `for(i in 1:20) plot(df[,i], main=colnames(df)[i])`

Session if+for Recap

- `if(TRUE) message("Yo!") else message("Dude...")`
- `ifelse(c(T,T,F,T), 7, else 7.3)` for vectors
- for loops repeat a block of code with a different `i` each time
- `for(i in 1:20) plot(df[,i], main=colnames(df)[i])`
- `for(cn in colnames(df)) plot(df[,cn], main=cn)`

Session if+for Recap

- `if(TRUE) message("Yo!") else message("Dude...")`
- `ifelse(c(T,T,F,T), 7, else 7.3)` for vectors
- for loops repeat a block of code with a different `i` each time
- `for(i in 1:20) plot(df[,i], main=colnames(df)[i])`
- `for(cn in colnames(df)) plot(df[,cn], main=cn)`
- for loops should not be called too often, as they are slow

Session if+for Recap

- `if(TRUE) message("Yo!") else message("Dude...")`
- `ifelse(c(T,T,F,T), 7, else 7.3)` for vectors
- for loops repeat a block of code with a different `i` each time
- `for(i in 1:20) plot(df[,i], main=colnames(df)[i])`
- `for(cn in colnames(df)) plot(df[,cn], main=cn)`
- for loops should not be called too often, as they are slow
- instead: vectorize, use lapply, write C++ code

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks

Course plan
- Session 2/4: Objects, file reading, Rstudio
- Session: if-else conditionals, for loops
- Session: functions and apply loops
- Session x: Recap all
- Feedback

Today's plan

1. Recap if else + for loop

Today's plan

1. Recap if else + for loop
2. Writing functions

Today's plan

1. Recap if else + for loop
2. Writing functions
3. optional: debugging functions

Recap if else I

1. Create a new R file and write `x <- 3.14` into it.
2. Now assign the absolute value of x to `y` by using conditional code execution (control structures).
3. `source` the complete document.
4. Test the result of y by setting `x` to a negative number in the script and running the document again.
5. BONUS: Make it also work for vectors like `x <- (-5):8` .
6. BONUS 2: Which R function gives the absolute value? How is it implemented?
   What's in github.com/wch/r-source -> src/main/names.c L278 and -> src/main/arithmetic.c L1315?

Recap if else I

1. Create a new R file and write `x <- 3.14` into it.
2. Now assign the absolute value of x to `y` by using conditional code execution (control structures).
3. `source` the complete document.
4. Test the result of y by setting `x` to a negative number in the script and running the document again.
5. BONUS: Make it also work for vectors like `x <- (-5):8` .
6. BONUS 2: Which R function gives the absolute value? How is it implemented?
   What's in github.com/wch/r-source -> src/main/names.c L278 and -> src/main/arithmetic.c L1315?

```
x <- -5:8
y <- ifelse(x<0, -x, x)
y

##  [1] 5 4 3 2 1 0 1 2 3 4 5 6 7 8
```

Recap if else II

```
(  requireNamespace("some_package", quietly=TRUE)  )

## [1] FALSE
```

invisibly returns TRUE if it succeeds loading a package. The outer brackets are for printing in the slides.

1. Use it to conditionally install the package quantmod (i.e. if it is not already available).

Recap if else II

```
(  requireNamespace("some_package", quietly=TRUE)  )

## [1] FALSE
```

invisibly returns TRUE if it succeeds loading a package. The outer brackets are for printing in the slides.

1. Use it to conditionally install the package quantmod (i.e. if it is not already available).

```
if(!requireNamespace("quantmod")) install.packages("quantmod")

## Loading required namespace:  quantmod
## Registered S3 method overwritten by 'quantmod':
##  method            from
##  as.zoo.data.frame zoo
```

Recap loops and conditions

Recap loops and conditions

Exercise 60: conditional loop

Repetition from last week with the built-in dataset `quakes` (locations of earthquakes off Fiji)

1. Create a multiple figure graph with `par(mfrow=c(2,3))`
2. With a for loop, create a histogram of each column in `quakes`. (One hist per panel).
3. Suppress the x and y axis label (using an empty label) in all plots (`xlab, ylab`).
4. Using conditional code execution, write a y axis label in the upper left panel (`title`).
5. BONUS1: Using `%in%`, how can you write a label in both left panels?
6. BONUS2: use the column name as titles for each graph.
7. BONUS3: use the same y axis limit (`ylim`) in each panel.
8. BONUS4: generally improve the graphic labeling, spaces etc.

conditional loop exercise solution

```
par(mfrow=c(2,3), mar=c(3,3,2,0.5), mgp=c(2,0.7,0), las=1)
for(c in colnames(quakes))
 {
 hist(quakes[,c], main=c, xlab="",ylab="", ylim=c(0,300), col="coral")
 if(c=="mag") title(ylab="Number of Obs")
 }   #  if(c %in% c("lat","mag")) title(ylab="Number of Obs")
```

Now let's start learning about functions

Jump to Functions

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks

Course plan
- Session 2/4: Objects, file reading, Rstudio
- Session: if-else conditionals, for loops
- Session: functions and apply loops
- Session x: Recap all
- Feedback

Floyd Recap

Download the recap tasks at FloydRecap.pdf

Floyd Recap Solutions

1.

```
setwd("S:/Dropbox/R/Project")
reed <- read.table("file.name", header=T, sep="\t", skip=2)
```

2. Spaces in column names, comments in the middle of data fields (use NA instead), tab-stop separated columns are not the safest option (semicolon ";" is better). `str(reed)` output:

```
data.frame, n observations in 4 variables (rows, columns)
$Site          : factor with n levels ...
$moisture      : factor with 3 levels "wet","moist",...: 1 2 2...
$plant.height  : int 154 120 173 ...
$stem_diameter: num 1.80.95 2.6 ...
```

3. see slide 111. `c(pi,"pi")` gives a vector with charstrings, number gets converted in order of coercion. You cannot compute with a characterstring, but you could run `as.numeric(c(pi,"pi")[1])*1.8`.

Floyd Recap Solutions

4. join "reed" with second dataset "chem":

```
merge(reed, chem, by.x="Site", by.y="columname", all=TRUE)
```

5.

```
plot(reed$chem, reed$growth)
```

6.

```
par(mfrow=c(2,3), mar=c(2.9,3,1,0.5), mgp=c(2.1,0.7,0), las=1)
plot(a,b)
plot(C,D, col="pink")
...
```

7-9.

```
par(mfrow=c(4,4), mar=rep(0,4), oma=c(3,5,2,1), las=1)
for(i in 1:16)
  {plot(rnorm(20), type="l", yaxt="n", xaxt="n")
  if( i %in% c(1,5,9,13) ) axis(side=2)
  }
title(ylab="hi there", outer=TRUE, cex.lab=2)
title(main="I'm cool", outer=TRUE)
box("outer", col="blue", lwd=5)
```

Floyd Recap Solutions

Outline

R course info
1. One-Session-Intro
2. Objects
3. "Real" data
4. Plotting
5. Character Operations
6. Conditions & loops
7. Functions & packages
8. Distributions
9. Statistics
10. Time series handling and analysis
11. Spatial data and GIS functionality
12. Final tasks

Course plan
- Session 2/4: Objects, file reading, Rstudio
- Session: if-else conditionals, for loops
- Session: functions and apply loops
- Session x: Recap all
- Feedback

Feedback

Please fill out the feedback form at

bit.ly/feedbackR

(it only takes a few minutes and helps to improve the course)

Thanks!