```
r[[rg_mm$max]] <- terra::extend(r[[rg_mm$max]], ext.combi)

# shift the smaller one over to the other side
ext.min <- terra::ext(r[[rg_mm$min]])
ext.min <- terra::ext(as.numeric(cc.xmin - min(rg)), cc.xmin, ext.min[3], ext.min[4])

terra::ext(r[[rg_mm$min]]) <- ext.min

# extent the smaller one too, resample to larger one
r[[rg_mm$min]] <- terra::extend(r[[rg_mm$min]], ext.combi)
r[[rg_mm$min]] <- terra::resample(r[[rg_mm$min]], r[[rg_mm$max]])

# fuse rasters over grid end
r <- terra::merge(r[[1]], r[[2]])
```

EAGLE M.Sc., MB2 2023/24

Introduction into programming using R

Part II: Concepts, code & practical exercises

Short recap

Last week and today morning, you have learned how to

- update a github fork by *syncing* it via GitHub (web browser)

Short recap

Last week and today morning, you have learned how to

- update a github fork by *syncing* it via GitHub (web browser)
- update any git repository from an upstream repository using *rebase* (terminal)

Short recap

Last week and today morning, you have learned how to

- update a github fork by *syncing* it via GitHub (web browser)
- update any git repository from an upstream repository using *rebase* (terminal)
- read lines from a file

Short recap

Last week and today morning, you have learned how to

- update a github fork by *syncing* it via GitHub (web browser)
- update any git repository from an upstream repository using *rebase* (terminal)
- read lines from a file
- read tabular data

Short recap

Last week and today morning, you have learned how to

- update a github fork by *syncing* it via GitHub (web browser)
- update any git repository from an upstream repository using *rebase* (terminal)
- read lines from a file
- read tabular data
- use functions and pass things to them through arguments

Short recap

Last week and today morning, you have learned how to

- update a github fork by *syncing* it via GitHub (web browser)
- update any git repository from an upstream repository using *rebase* (terminal)
- read lines from a file
- read tabular data
- use functions and pass things to them through arguments
- index and manipulate a vector

Short recap

Last week and today morning, you have learned how to

- update a github fork by *syncing* it via GitHub (web browser)
- update any git repository from an upstream repository using *rebase* (terminal)
- read lines from a file
- read tabular data
- use functions and pass things to them through arguments
- index and manipulate a vector
- convert a vector into a matrix

Short recap

Last week and today morning, you have learned how to

- update a github fork by *syncing* it via GitHub (web browser)
- update any git repository from an upstream repository using *rebase* (terminal)
- read lines from a file
- read tabular data
- use functions and pass things to them through arguments
- index and manipulate a vector
- convert a vector into a matrix
- replace, match character patterns

Short recap

Last week and today morning, you have learned how to

- update a github fork by *syncing* it via GitHub (web browser)
- update any git repository from an upstream repository using *rebase* (terminal)
- read lines from a file
- read tabular data
- use functions and pass things to them through arguments
- index and manipulate a vector
- convert a vector into a matrix
- replace, match character patterns
- count unique elements of a vector

Tasks from last week:

(1) Create a vector containing all languages mentioned,
(2) create a wordcloud plot using that vector

Solution

(1) Create a vector containing all languages mentioned,
(2) create a wordcloud plot using that vector

Any questions regarding these topics?

Agenda

- Background: Modes, structures and indexing
- Practical: Reading, exploring, plotting data using R and the course's example ENTSO-E enegery dataset

Follow-up and joint task:

Types, structures and indexing

Types and structures

There are four main *modes* in which R can represent *scalars* (individual values):

- character (for strings)
- logical
- double (for floating points)
- integer (integer + double are collectively referred to as "numeric" in R)
- complex
- raw

Types and structures

There are four main *modes* in which R can represent *scalars* (individual values):

- character (for strings)
- logical
- double (for floating points)
- integer (integer + double are collectively referred to as "numeric" in R)
- complex
- raw

```
typeof("a")
# > [1] "character"

typeof(TRUE)
# > [1] "logical"

typeof(1.3)
# > [1] "double"
```

Types and structures

Scalars can be bound together to form *atomic vectors* that

- can represent only one *type* at a time
- can hold a multitude of values
- can carry *attributes* which are used to e.g. store dimensions, names and other things to form complex *structures* or *classes*

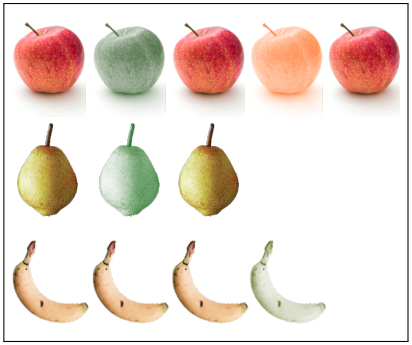
Structures

There are four main *structures* in R that are build from *atomic vectors* (and are the foundation for many *classes* of objects):

- *vectors* themselves
- *matrices*
- *lists*
- *data.frames*

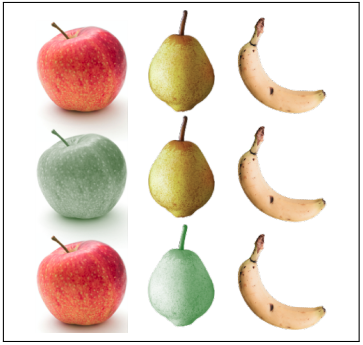
Which structures are displayed here?

1



2

3



4

Indexing

5 If "cabinet" would be a data class, you would index its drawes – *pexels.com*

Structures and indexing

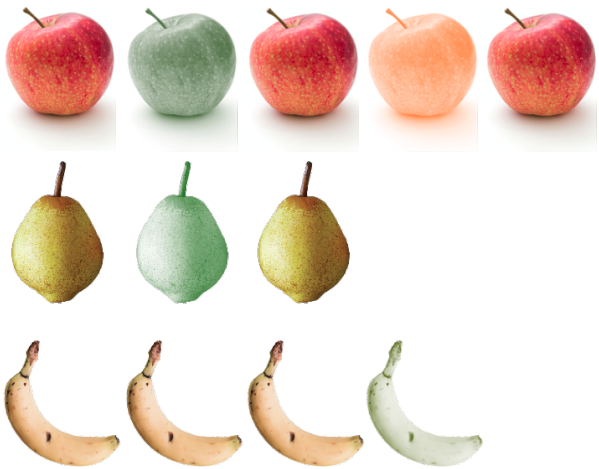
6 What kind of structure is represented here? – *own illustration*

Structures and indexing

7 A *vector* of a single type and its index – *own illustration*

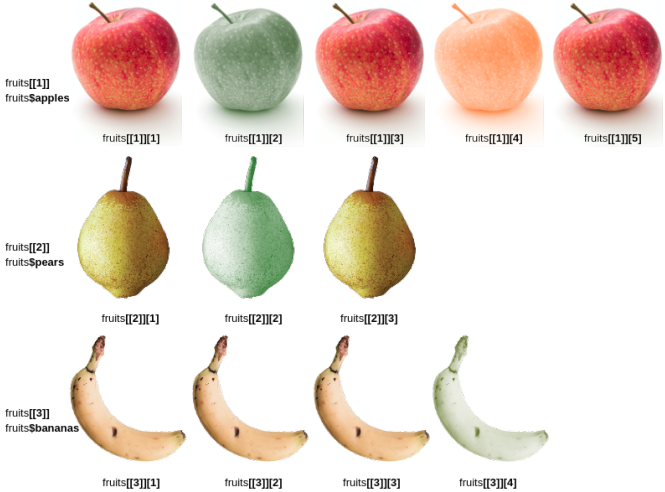
8 A *vector* of a single type and its index – *own illustration*

Structures and indexing



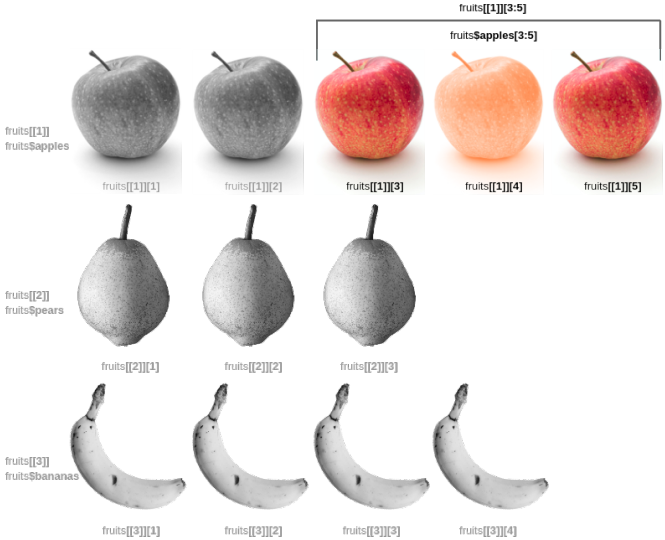
9 What kind of structure is represented here? – *own illustration*

Structures and indexing



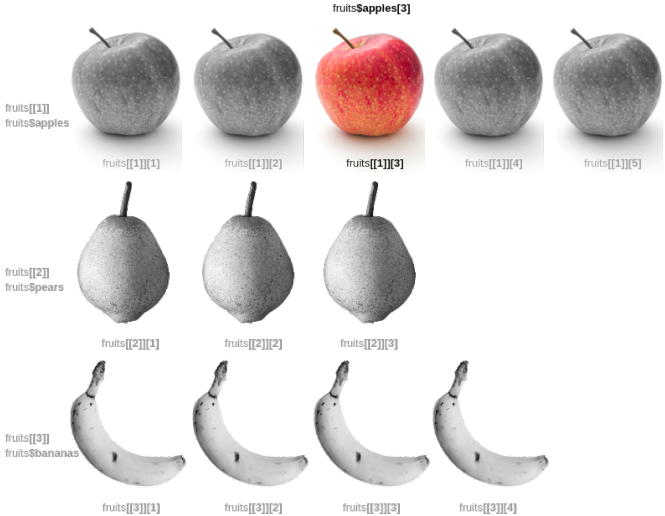
10 A *list* of vectors of different type and its index – *own illustration*

Structures and indexing



11 A *list* of vectors of different type and its index – *own illustration*

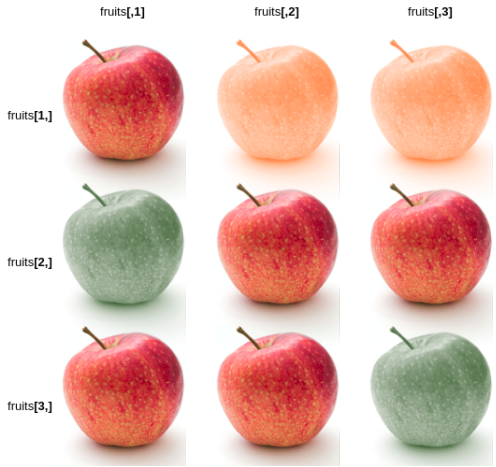
Structures and indexing



12 A *list* of vectors of different type and its index – *own illustration*

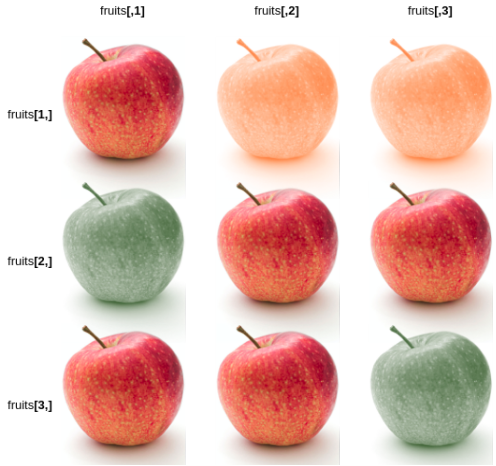
Structures and indexing

13 What kind of structure is illustrated here? – *own illustration*



14 A *matrix* of a single type and its index – *own illustration*

Structures and indexing



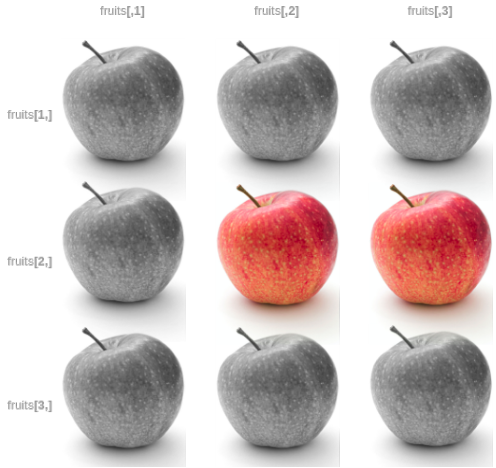
15 A *matrix* of a single type and its index – *own illustration*

Creating a *matrix*:

```
fruits <- matrix(c(1, 2, 1, 3, 1, 1, 3, 1, 2),
                 ncol = 3)

# or
fruits <- rbind(c(1, 2, 1),
                c(3, 1, 1),
                c(3, 1, 2))
```

Structures and indexing

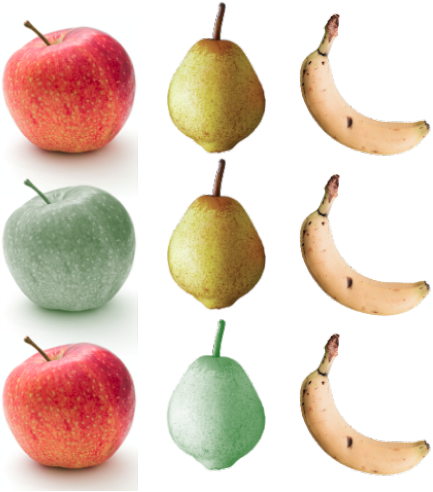


16 A *matrix* of a single type and its index – *own illustration*

Indexing a *matrix*:

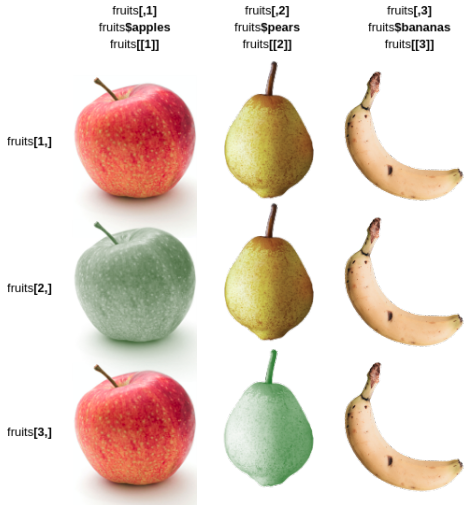
```
1 # index
2 fruits[2,2:3]
```

Structures and indexing



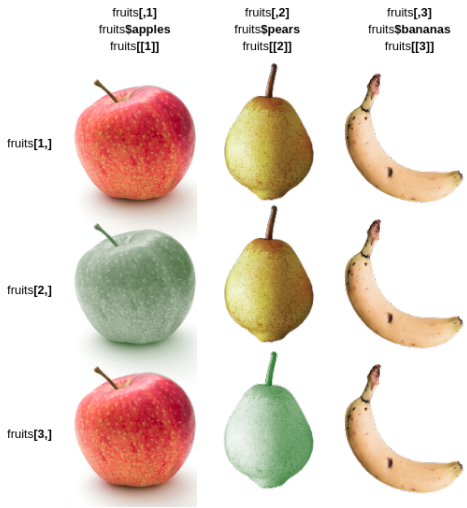
17 What kind of structure is illustrated here? – *own illustration*

Structures and indexing



18 A *data.frame* of multiple types and its index – *own illustration*

Structures and indexing



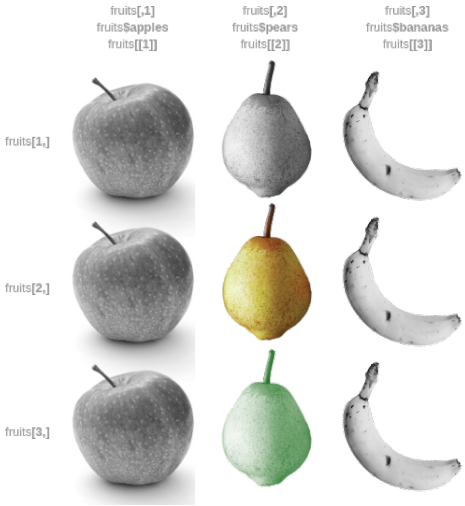
19 A *data.frame* of multiple types and its index – *own illustration*

Naming a *data.frame*:

```
fruits <- data.frame(apples = c(1, 2, 1),
                     pears = c("a", "a", "b"),
                     bananas = c(TRUE, TRUE, TRUE))

# or name the columns later
fruits <- data.frame(c(1, 2, 1),
                     c("a", "a", "b"),
                     c(TRUE, TRUE, TRUE))
colnames(fruits) <- c("apples", "pears", "bananas")
```

Structures and indexing



20 A *data.frame* of multiple types and its index – *own illustration*

Indexing a *data.frame*:

```
1 fruits[2:3,2]
2 fruits$apples[2:3]
3 fruits[[2]][2:3]
```

Follow-up on reading tabular data:

- Reading datasets created on other systems with different software and different data philosophies is a **non-trivial** problem!
- Thus many different and new packages emerge for reading data (even if it is "only" CSVs)
- Some additional usefull packages are
  - *readr* (tidyverse),
  - *rio* (especially if you have encoding issues),
  - *data.table* (especially for high-speed reading)
- If you feel already very confident with reading tabular using base R, try to utilize them in the upcoming task!

Task: Dataset

Dataset: Energy production across Europe per Power Unit

- All power units directly supplying to the European grid are represented
- Temporal resoultion by at least the hour
- Actual production output (how much did the plant produce at the time) vs. installed output
- Categories by location, production type etc.

Task: Reading, indexing and exploring a tabular dataset

Aim: Read, index, analyze.

1. Make sure your local MB12 repository is in sync with the upstream repository.
2. Try to read the file *day2_data_energy_prod_EU_2020-08-03_2020-08-09.csv* into R using a function for reading CSVs.
3. Explore/filter the dataset. Check out columns and what they tell. Try to understand the dataset.

Task: Reading, indexing and exploring a tabular dataset

Aim: Read, index, analyze.

1. Make sure your local MB12 repository is in sync with the upstream repository.
2. Try to read the file *day2_data_energy_prod_EU_2020-08-03_2020-08-09.csv* into R using a function for reading CSVs.
3. Explore/filter the dataset. Check out columns and what they tell. Try to understand the dataset.

Extra for those interested/if you have time:

- Get an idea of what sort of (simple) statistics/analysis you could do on one or multiple of the variables. Try to do get some information out of the data!
- Try to create plots that tell us something about the dataset.
- Save your results (plots, scripts etc.) to your local git repo and push them to your GitHub repo.

Live coding

Exploring the *ENTSO-E dataset* using R

Any questions?

Feedback

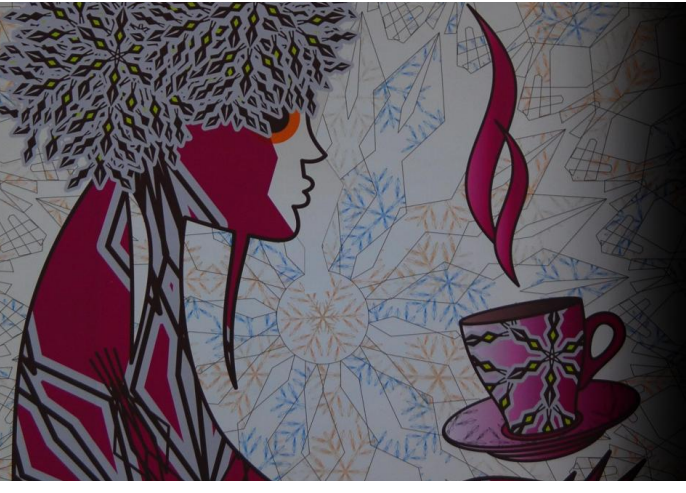
Anonymous feedback via slido.com, event code **#mb2training**.

What we would like to know from you:

- Is the speed ok?
- Is the session structure ok?
- Is there any content you'd like to be covered?
- Is it boring or not?
- Is it helpful or not?
- Is the presentation style ok?
- Anything else?

Go to slido.com & join #mb2training

Please give feedback using the feedback box.



Coffee time – see you next week