University of Zurich[UZH]

Department of Geography

GEO 812
Getting started with R for Spatial Analysis

Session 2: Data types and data handling

Peter Ranacher

September 2019

Learning objectives

You are able to

- identify objects, assignments, values and functions in R code
- name and explain R's most important data types
- import external data to R
- find help in case of problems

A simple example:

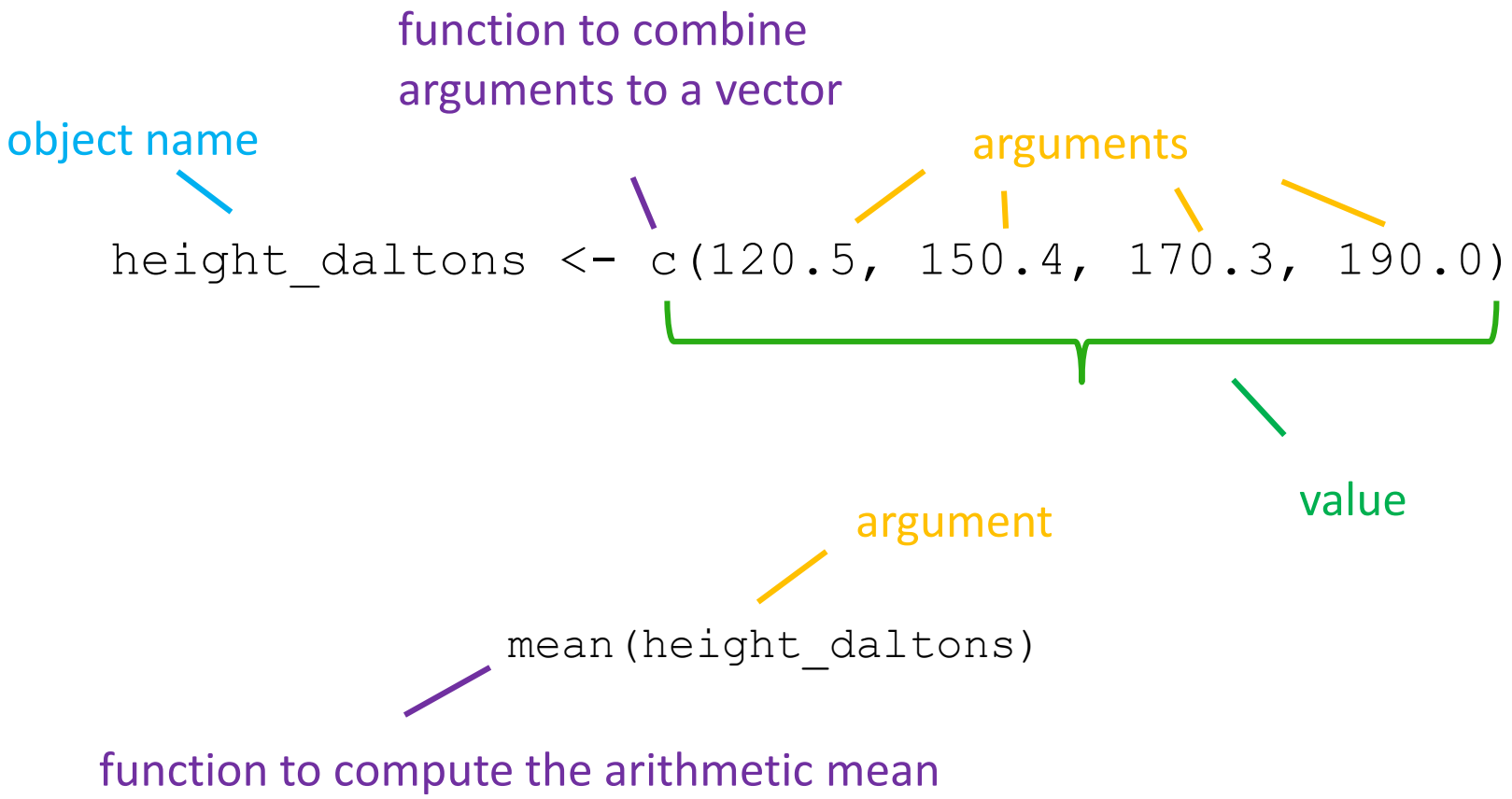
The four Dalton brothers are 120.5, 150.4, 170.3 and 190.0 cm tall.

What is the average height of the Daltons?

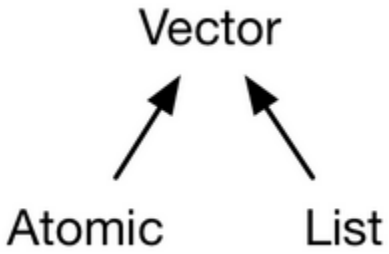
```
height_daltons <- c(120.5, 150.4, 170.3, 190.0)

mean(height_daltons)
```

Basic ingredients: objects, values, functions and arguments



Vectors



Atomic vectors are sequences of data elements of the same type.

Lists are recursive vectors. They can contain atomic vectors and other lists.

All vectors have

- a type
- and a length

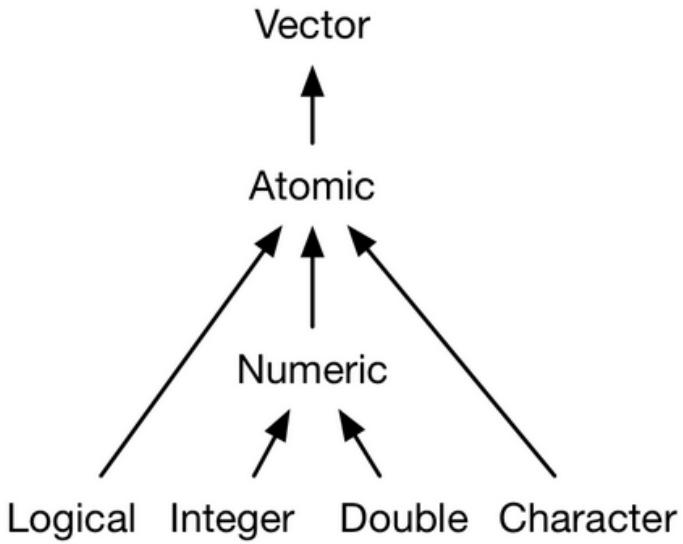
Try `typeof(height_daltons)` and `length(height_daltons)`

Some vectors have attributes (more on that later!)

All variables we have defined so far are vectors!

Atomic Vectors

four basic types



Joe	Jack	William	Averell

Character `names_daltons <- c("Joe", "William", "Jack", "Averell")`

Double `height_daltons`

Integer `as.integer(height_daltons)` or `c(120L, 150L, 170L, 190L)`

Logical `is_smart_daltons <- c(TRUE, TRUE, TRUE, FALSE)`

Testing vectors

Check if an object is a vector (vector, not atomic vector!)….

```
is.vector(names_daltons)
```

…. and if an atomic vector is of type character, numeric, logical, double or integer:

```
is.character(names_daltons)
is.numeric(names_daltons)
is.logical(is_smart_daltons)
is.double(height_daltons)
is.integer(height_daltons)
```

Coercion

all elements of an atomic vector must be of the same type

if not they will be forced into the most flexible mode

```
a <- c(TRUE, "hello", 1)
typeof(a)
```

Coercion =

Subsetting atomic vectors

Subset with a numeric vector containing only integers

`names_daltons[1]`	`[n]`	get the n^{th} element
`names_daltons[2:3]`	`[m:n]`	get all elements from m to n
`names_daltons[c(1, 4)]`	`[c(m, n)]`	get element m and n

Subset with a logical vector

```
index <- height_daltons > 120
height_daltons[index]
```

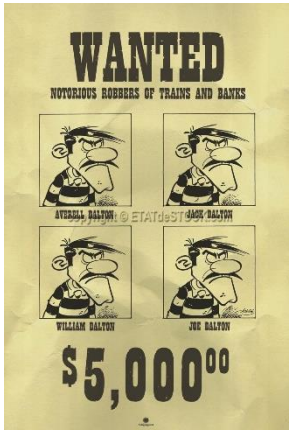
Assigning values to atomic vectors

Create an empty atomic vector of given length and given type

```
reward_daltons <- vector(mode = "integer", 2)
```

Use indexing to assign values to the atomic vector

```
reward_daltons[1] <- 5000L
reward_daltons[2:3] <- 2000L
reward_daltons[4] <- 1000L
```



`mode = "numeric"` ??
I thought it was type!

R was developed by a large community.
It is not always consistent.

Some useful functions for atomic vectors

Sum of all elements of an atomic vector

```
sum(reward_daltons)
```

Product of all elements of an atomic vector

```
prod(is_smart_daltons)
```

A product of a logical value?
TRUE = 1, FALSE = 0!

Sort the elements of an atomic vector

```
sort(names_daltons)
```

Minimum, maximum and range

```
min(reward_daltons)
max(reward_daltons)
range(reward_daltons)
```

Recycling

The shorter vector is repeated, or recycled, to the same length as the longer vector.

```
height_daltons + c(10, 10, 10, 10)
height_daltons + 10
height_daltons + c(10, 20)
```

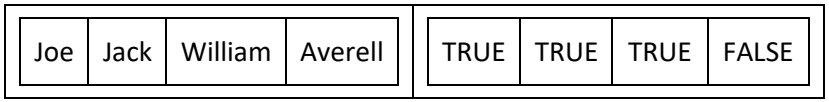
Recycling is not always possible

```
height_daltons + c(10, 20, 30)
```

Be careful when applying arithmetic operations to vectors!

Lists

- Recursive vectors
- ordered collections of elements
- allow for combining different atomic vectors and even lists

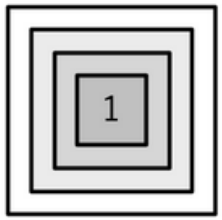


Collect some data in a single list:

```
daltons_list <- list(height_daltons, names_daltons)
```

Putting lists in lists

```
meta_list <- list(list(list(1)))
```



Try: `typeof(daltons_list)`
`is.vector(daltons_list)`

Naming vectors

Name an atomic vector during creation

```
c("Joe" = 5000L, "Jack" = 2000L, "William" = 2000L, "Averell" = 1000L)
```

Assign names after the creation

```
names(reward_daltons) <- c("Joe", "Jack", "William", "Averell")
```

The same works for lists

```
daltons_list <- list(height = height_daltons, names = names_daltons)
names(daltons_list) <- c("height", "names")
```

Try: `attributes(reward_daltons)`
and `attributes(height_daltons)`

Subsetting lists

Extract a sub-list (the result is a list!)

```
daltons_list[2]
daltons_list[1:2]
```

Extract single components from a list

```
daltons_list[[2]]
daltons_list[["names"]]
daltons_list$names
```

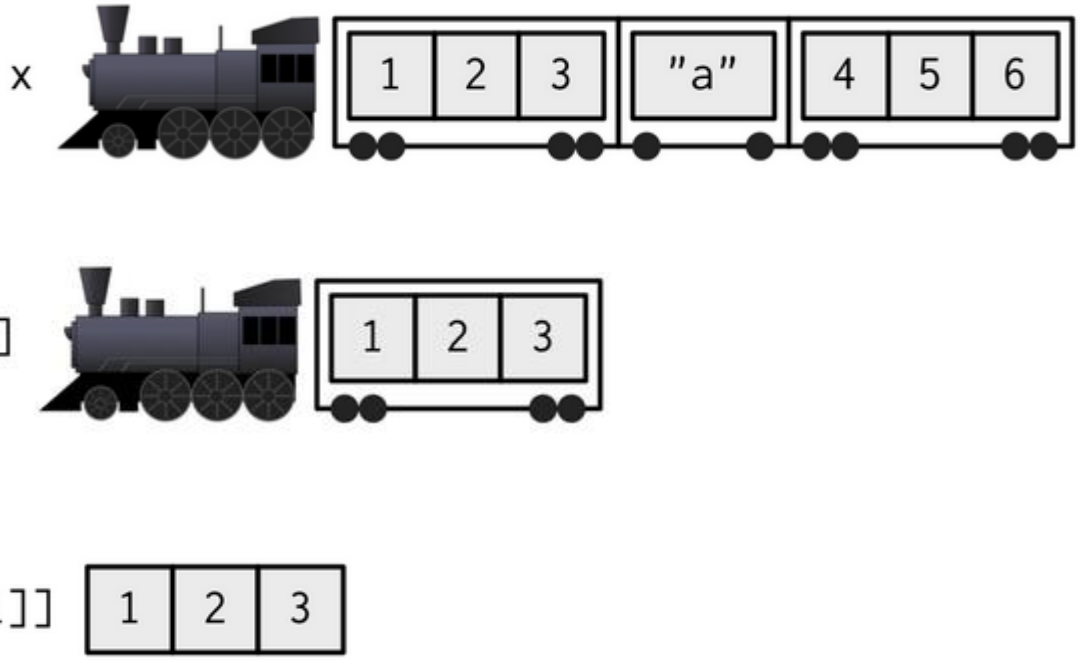
Subsetting the subset

```
daltons_list[[2]][1]
```

What's wrong with `daltons_list[2][1]`?

Subsetting lists explained visually

The difference between `[[` and `[`



Exercise 3:

1. a. Define:

```
a <- c(49, 16, 4)
b <- c(10, 3, 5)
c <- c(12, 9)
d <- TRUE
```

b. Calculate:

```
sqrt(a)
a + b
a + c
a + d
```

c. Interpret your results!

2. You want to collect the daily high temperature for three Swiss cities for August 25. Unfortunately, your weather app works in degrees Fahrenheit (° F) and not in degrees Celsius (° C). According to your app, it had 70 ° F in Zurich, 75 ° F in Berne and 80 ° F in Lausanne.

a. Store your observations as an atomic vector called `temp_F`.

b. Transform the degrees Fahrenheit to degrees Celsius and store your results in `temp_C`. The following formula might help you: ° C = (° F - 32) * 5/9. Name your vector elements. Use the cities as names.

c. What are the maximum and the minimum temperature in `temp_C`?

Vector attributes

Metadata about vectors

Names

are used to name elements of a vector

Dimension (dims)

makes vectors multi-dimensional (2d matrix, nd array)

Class

from object oriented programming (OOP)

defines a type of object with methods and attribute associated to it

Get all attribute of a vector

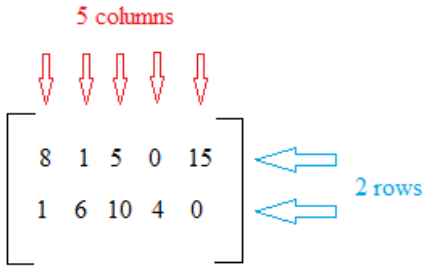
```
attributes(reward_daltons)
```

Get a specific attribute of a vector

```
attr(reward_daltons, "names")
```

Matrices

- vectors of dimension 2
- data arranged in a two-dimensional rectangular layout
- consist of rows and columns



Create an empty matrix of 3 rows and 2 columns:

```
matrix(data = NA, nrow = 3, ncol = 2)
```

number of columns

matrix function

number of rows

Matrices (continued)

The Daltons robbed the bank in Dead Ox Gulch twice and the saloon once. They robbed the bank in Desert City three times and the saloon four times. In Frontier City, they have neither robbed the bank nor the saloon.

In matrix notation?

```
robberies_daltons <- matrix(data = c(2, 1, 3, 4, 0, 0),
                            ncol =2, nrow = 3, byrow = T)
```

How to fill the data into the matrix?

Matrix rows and columns can be named

```
colnames(robberies_daltons) <- c("bank", "saloon")
rownames(robberies_daltons) <- c("Dead Ox Gulch", "Desert City",
                                 "Frontier City")
```

Subsetting matrices

Use element position or names for indexing

Get the element in row one and two

```
robberies_daltons[1, 2]
robberies_daltons["Dead Ox Gulch", "saloon"]
```

Get all elements in row three

```
robberies_daltons[3, ]
robberies_daltons["Frontier City", ]
```

Zeilen zuerst, Spalten später
(first the rows,
then the columns)!

Get all elements in column two

```
robberies_daltons[, 2]
robberies_daltons[, "saloon"]
```

Augmented vectors

- are built on atomic vectors
- have a **class**, which makes them behave differently

data frames (tibbles) are augmented list

factors are augmented integers

dates and **date-times** are augmented double

Data frames (tibbles)

- a list where all elements have the same length
- share properties of a matrix and a list

Create a data frame from existing data

```
daltons_df <- data.frame(height = height_daltons,
                         reward = reward_daltons)
```

Subsetting the "matrix way"

```
daltons_df[, "height"]
daltons_df[1 , 2]
```

Try `attributes(daltons_df)`

Subsetting the "list way"

```
daltons_df$reward
```

`cbind` and `rbind`

Combine vectors by column (`cbind`) or rows (`rbind`)

Combine data frames by column:

```
cbind(daltons_df, data.frame(is_smart_daltons))
```

Combine atomic vectors by rows:

```
rbind(height_daltons, is_smart_daltons)
```

Structure needs to match! Otherwise error or coercion!

Factors

- vectors that take a fixed set of possible values
- represent categorical data
- build on integers!

Try `attributes(danger_daltons)`

```
factor(x, levels)
```

possible categories

```
danger_daltons <- factor(c("very dangerous", "dangerous",
                           "dangerous", "harmless"),
                         levels = c("ultra violent", "very dangerous",
                                    "dangerous", "harmless"))
```

Exercise 4:

1. Your colleague collected precipitation data for August 25 for Zurich, Berne and Lausanne:

```
prec <- c(0, 100, 0)
```

   a. Use the commands `c_matr <- cbind(temp_C, prec)` and `r_matr <- rbind(temp_C, prec)` to combine the precipitation data with the temperature data in a matrix. What happens to the names of the elements? Try to assign them to the matrix.

   b. Compare `c_matr` and `r_matr`! What is the difference? Which one would you prefer for your analysis and why?

   c. Convert your preferred choice into a data frame named `weather`.

2. You find out that your app also collects a rough estimate for when the daily high was recorded. You store this information as a factor named `time`.

```
time <- factor(c("afternoon", "afternoon", "noon"),
               levels = c("morning", "noon", "afternoon", "night"))
```

   Add `time` as a column to `weather`. Change the time for Zurich to `"evening"`. Explain what happens!

Read tables and csv

Read ASCII text files

```
read.table(file = "your.txt", header = T)
```

Read CSV

```
stats_geo812 <- read.csv(file = "data/stats_geo812.csv",
                         header = T, sep = ",")
```

- ALWAYS view the file first before you read it into R!
- For directories use the forward slash (/), not the backslash (\)
  EVEN ON WINDOWS

Troubleshooting and Organization

- Organize your R project
- Find help
- Packages, data sets and vignettes
- Tips and tricks on how to write good code in R

Mini conference

- Four topics <-> four groups
- Each group

  prepares a topic (15 minutes, handouts will be provided)
  and then presents the topic (max 5 minutes per group)

Revisiting the learning objectives

You are able to

- identify objects, assignments, values and functions in R code
- name and explain R's most important data types
- import external data to R
- find help in case of problems