R Data

Data types and classes

Matilde Trevisani

R objects

R objects

- “Everything that exists in R is an object”.
- Un oggetto è una struttura di dati specializzata
- Tipi o strutture base sono
  - *Vectors*
  - *Lists*
  - *Language objects*
  - *Expression objects*
  - *Function objects*
  - NULL
  - Environments
  - . . .
  - . . .

Data types

Objects types in R

Ognuno di questi oggetti può essere di vario **tipo**

Per conoscere il tipo di un oggetto possiamo usare la funzione `typeof` (o anche `mode` che può differire in qualche singolo caso)

Ad esempio

- *Vectors* include types NULL, logical, integer, double, complex, character, list, and raw.
- *Functions* include types closure (regular R functions), special (internal functions), and builtin (pri.mitive functions).
- etc

Vectors

Noi ci concentriamo ora sui **vettori**, la più importante struttura di dati in R.

Una lettura chiara e completa nel capitolo a https://r4ds.had.co.nz/vectors.html.

Un vettore è una serie di valori, ognuno dei quali è uno **scalare**.

Vedremo che partendo dalle strutture base e arricchendole con determinati **attributi** ricostruiremo strutture di dati in R via via più articolate e complesse (*augmented vectors*).

Atomic vectors

Cominceremo con i cosiddetti *atomic vectors*, quelli che comunemente chiamiamo vettori, contraddistinti dal fatto di essere **omogenei**, cioè di essere costituiti da valori tutti dello stesso tipo.

Andiamo a vedere quali sono i tipi di vettori considerando dapprima gli scalari (vettori di lunghezza 1!), i singoli valori che li costituiscono.

Tipi di vettori

Ci sono quattro tipi base, e poi due meno usati.

- **logical**
- **integer**
- **double**
- **character**
- e qualcun altro (**complex**, **raw**), ma che tralasceremo.

Sono in ordine di complessità.

Integer e double (numeri reali) sono insieme noti come **numeric**.

Scalari o singoli valori di un vettore

I singoli valori (scalari) sono nei quattro tipi principali:

- Logicals o Boolean values sono TRUE e FALSE
  - the term Boolean means a result that can only have one of two possible values: true or false
- Integers sono valori come -1, 0, 2, 4092 e devono essere scritti seguiti da una `L`.
- Doubles sono un più ampio insieme di valori che contiene sia gli integers ma anche fractions e decimal values like -24.932 and 0.8.
  - Doubles can be specified in decimal (0.1234), scientific (1.23e4), or hexadecimal (0xcafe) form.
- Characters or character strings are text such as “Hello” e “Come stai oggi?”
  - characters are often denoted with the quotation marks around them, either single quotes (') or double quotes (").

Ci sono poi valori speciali, `NA` per logicals, integers and doubles. Inoltre altri tre, `NaN`, `Inf` e `-Inf`, solo per i doubles.

Logical & character

Per conoscere il tipo di dato possiamo usare la funzione `typeof` o anche `mode` (differiscono in qualche singolo caso)

logical - boolean values `TRUE` and `FALSE`

```r
typeof(TRUE)
```

```
## [1] "logical"
```

```r
mode(TRUE)
```

```
## [1] "logical"
```

character - character strings

```r
typeof("hello")
```

```
## [1] "character"
```

```r
mode("hello")
```

```
## [1] "character"
```

Double & integer

double - floating point numerical values (default numerical type)

```
typeof(1.335)
```

```
## [1] "double"
```

```
typeof(7)
```

```
## [1] "double"
```

```
mode(1.335) ; mode(7)
```

```
## [1] "numeric"

## [1] "numeric"
```

integer - integer numerical values (indicated with an `L`)

```
typeof(7L)
```

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

```
typeof(1:3)
```

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

```
mode(7L) ; mode(1:3)
```

```
## [1] "numeric"

## [1] "numeric"
```

Converting between types

with intention...

```
x <- c(TRUE, FALSE)
x
```

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

```
typeof(x)
```

```
## [1] "logical"
```

```
y <- as.numeric(x)
y
```

```
## [1] 1 0
```

```
typeof(y)
```

```
## [1] "double"
```

Converting between types

with intention…

```r
x <- 1:3
x
```

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

```r
typeof(x)
```

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

```r
y <- as.character(x)
y
```

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

```r
typeof(y)
```

```
## [1] "character"
```

Concatenation

Vectors can be constructed using the `c()` function (*combine*).

```
c(1, 2, 3)
```

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

```
c("Hello", "World!")
```

```
## [1] "Hello"  "World!"
```

```
c(c("hi", "hello"), c("bye", "goodbye"))
```

```
## [1] "hi"      "hello"   "bye"     "goodbye"
```

Converting between types

without intention...

R convertirà tra diversi tipi quando tipi differenti di dati sono concatenati in un vettore, e questo può generare risultati non voluti!

```r
c(1, "Hello") ; typeof(c(1, "Hello"))
```

```
## [1] "1"     "Hello"
```

```
## [1] "character"
```

```r
c(FALSE, 3L) ; typeof(c(FALSE, 3L))
```

```
## [1] 0 3
```

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

```r
c(1.2, 3L) ; typeof(c(1.2, 3L))
```

```
## [1] 1.2 3.0
```

```
## [1] "double"
```

```r
c(2L, "two") ; typeof(c(2L, "two"))
```

```
## [1] "2"   "two"
```

```
## [1] "character"
```

Explicit vs. implicit coercion

- **Explicit coercion** is when you call a function like `as.logical()`, `as.numeric()`, `as.integer()`, `as.double()`, or `as.character()`

- **Implicit coercion** succede quando si mescolano tipi differenti in un vettore ("vince" la classe più generica)

Special values

Special values

- `NA`: Not available
- `NaN`: Not a number
- `Inf`: Positive infinity
- `-Inf`: Negative infinity

```r
pi / 0
```

```
## [1] Inf
```

```r
0 / 0
```

```
## [1] NaN
```

```r
1/0 - 1/0
```

```
## [1] NaN
```

```r
1/0 + 1/0
```

```
## [1] Inf
```

NAs are special ❄️s

R uses NA to represent **missing values** in its data structures.

```r
x <- c(1, 2, 3, 4, NA)
```

```r
mean(x)
```

```
## [1] NA
```

```r
mean(x, na.rm = TRUE)
```

```
## [1] 2.5
```

```r
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
##    1.00    1.75    2.50    2.50    3.25    4.00       1 
```

NAs are logical

```
typeof(NA)
```

```
## [1] "logical"
```

Mental model for NAs

- Unlike NaN, NAs are genuinely unknown values
- But that doesn't mean they can't function in a logical way
- Let's think about why NAs are logical...

> Why do the following give different answers?

→ See next slide for answers...

```
# TRUE or NA
TRUE | NA
```

```
## [1] TRUE
```

```
# FALSE or NA
FALSE | NA
```

```
## [1] NA
```

- NA is unknown, so it could be TRUE or FALSE

- TRUE | NA

```r
TRUE | TRUE  # if NA was TRUE
```

```
## [1] TRUE
```

```r
TRUE | FALSE # if NA was FALSE
```

```
## [1] TRUE
```

- FALSE | NA

```r
FALSE | TRUE  # if NA was TRUE
```

```
## [1] TRUE
```

```r
FALSE | FALSE # if NA was FALSE
```

```
## [1] FALSE
```

- Doesn't make sense for mathematical operations
- Makes sense in the context of missing data

Why should you care about data types?

Example: Cat lovers

A survey asked respondents their name and number of cats. The instructions said to enter the number of cats as a numerical value.

```
cat_lovers <- read_csv("data/cat-lovers.csv")
```

```
## # A tibble: 60 × 3
##   name           number_of_cats handedness
##   <chr>          <chr>          <chr>     
## 1 Bernice Warren 0              left      
## 2 Woodrow Stone  0              left      
## 3 Willie Bass    1              left      
## 4 Tyrone Estrada 3              left      
## 5 Alex Daniels   3              left      
## 6 Jane Bates     2              left      
## # ℹ 54 more rows
```

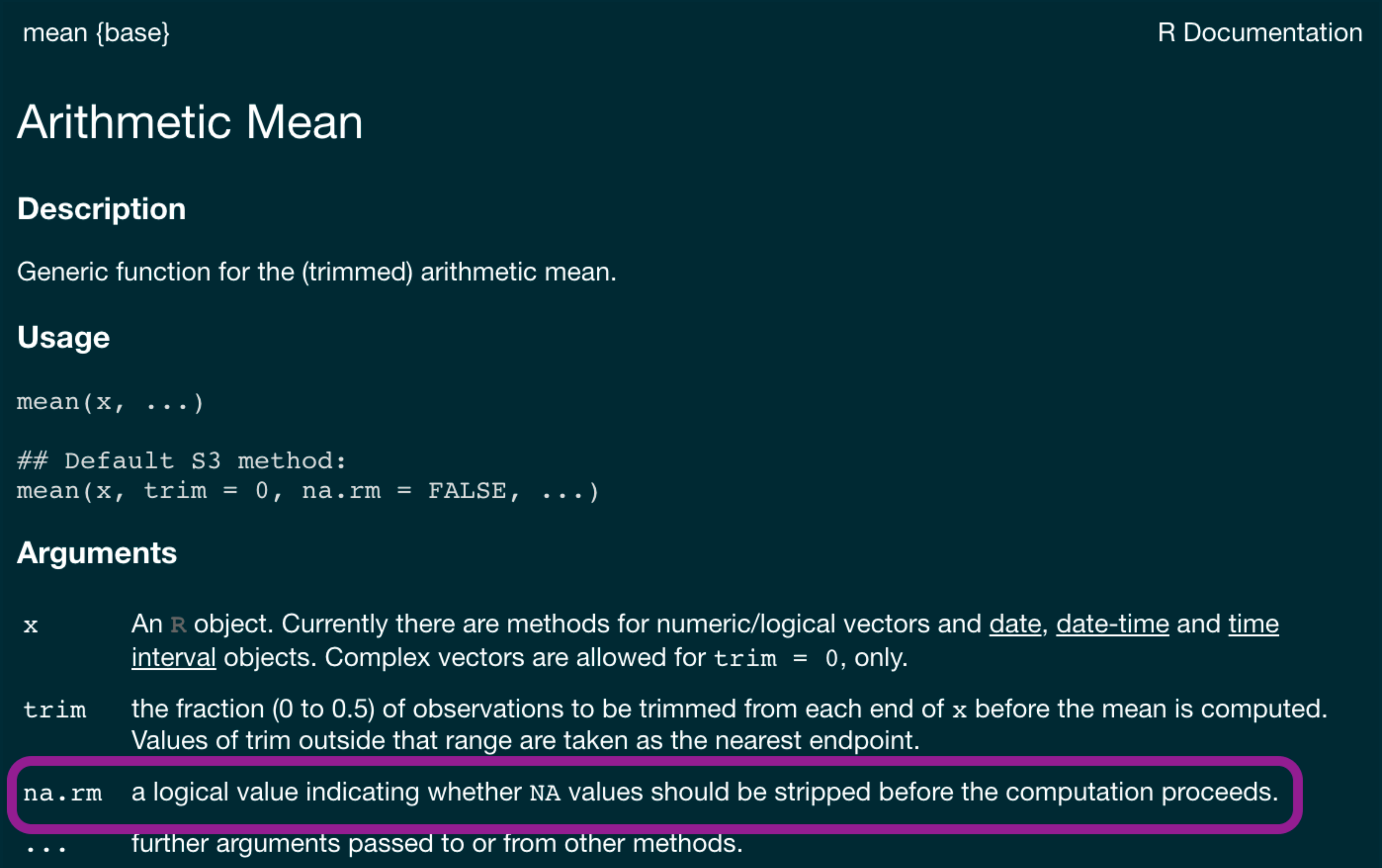
Oh why won't you work?!

```
cat_lovers %>%
  summarise(mean_cats = mean(number_of_cats))
```

```
## Warning: There was 1 warning in `summarise()`.
## ℹ In argument: `mean_cats = mean(number_of_cats)`.
## Caused by warning in `mean.default()`:
## ! l'argomento non è numerico o logico: si restituisce NA
```

```
## # A tibble: 1 × 1
##   mean_cats
##       <dbl>
## 1        NA
```

```r
?mean
```



mean {base} R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

`x` An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.

`trim` the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.

`na.rm` a logical value indicating whether `NA` values should be stripped before the computation proceeds.

`...` further arguments passed to or from other methods.

Oh why won't you still work??!!

```r
cat_lovers %>%
  summarise(mean_cats = mean(number_of_cats, na.rm = TRUE))
```

```
## Warning: There was 1 warning in `summarise()`.
## ℹ In argument: `mean_cats = mean(number_of_cats, na.rm = TRUE)`.
## Caused by warning in `mean.default()`:
## ! l'argomento non è numerico o logico: si restituisce NA
```

```
## # A tibble: 1 × 1
##   mean_cats
##       <dbl>
## 1        NA
```

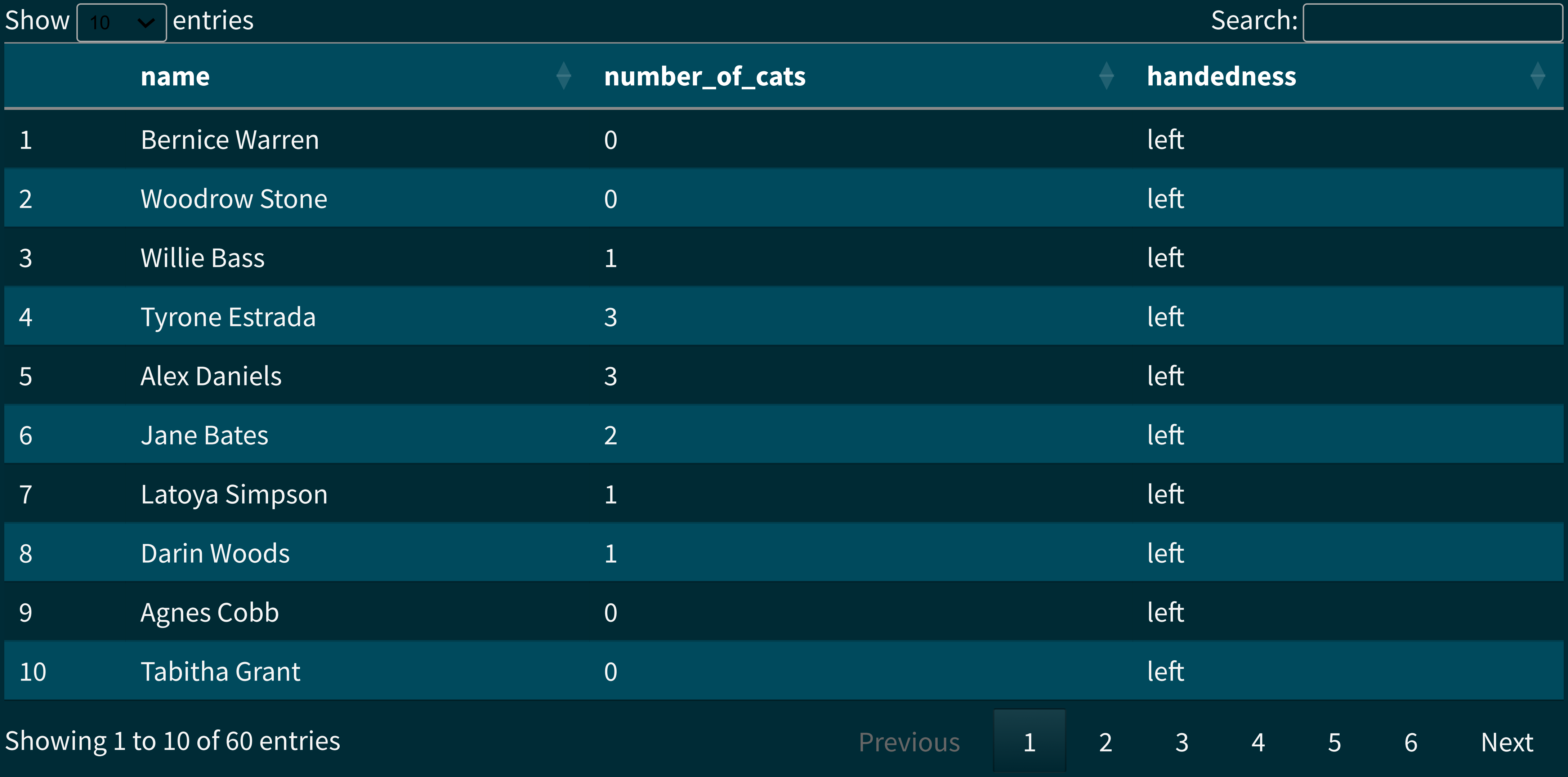
Take a breath and look at your data

> What is the type of the `number_of_cats` variable?

```
glimpse(cat_lovers)
```

```
## Rows: 60
## Columns: 3
## $ name           <chr> "Bernice Warren", "Woodrow Stone", "Will…
## $ number_of_cats <chr> "0", "0", "1", "3", "3", "2", "1", "1", …
## $ handedness     <chr> "left", "left", "left", "left", "left", …
```

Let's take another look



	name	number_of_cats	handedness
1	Bernice Warren	0	left
2	Woodrow Stone	0	left
3	Willie Bass	1	left
4	Tyrone Estrada	3	left
5	Alex Daniels	3	left
6	Jane Bates	2	left
7	Latoya Simpson	1	left
8	Darin Woods	1	left
9	Agnes Cobb	0	left
10	Tabitha Grant	0	left

Showing 1 to 10 of 60 entries

Sometimes you might need to babysit your respondents

```
cat_lovers %>%
  mutate(number_of_cats = case_when(
    name == "Ginger Clark" ~ 2,
    name == "Doug Bass"    ~ 3,
    TRUE                   ~ as.numeric(number_of_cats)
    )) %>%
  summarise(mean_cats = mean(number_of_cats))
```

```
## Warning: There was 1 warning in `mutate()`.
## ℹ In argument: `number_of_cats = case_when(...)`.
## Caused by warning:
## ! NA introdotti per coercizione

## # A tibble: 1 × 1
##   mean_cats
##       <dbl>
## 1     0.833
```

Always you need to respect data types

```r
cat_lovers %>%
  mutate(
    number_of_cats = case_when(
      name == "Ginger Clark" ~ "2",
      name == "Doug Bass"    ~ "3",
      TRUE                   ~ number_of_cats
      ),
    number_of_cats = as.numeric(number_of_cats)
    ) %>%
  summarise(mean_cats = mean(number_of_cats))
```

```
## # A tibble: 1 × 1
##   mean_cats
##       <dbl>
## 1     0.833
```

Now that we know what we're doing...

```r
cat_lovers <- cat_lovers %>%
  mutate(
    number_of_cats = case_when(
      name == "Ginger Clark" ~ "2",
      name == "Doug Bass"    ~ "3",
      TRUE                   ~ number_of_cats
      ),
    number_of_cats = as.numeric(number_of_cats)
    )
```

Moral of the story

- If your data does not behave how you expect it to, type coercion upon reading in the data might be the reason.
- Go in and investigate your data, apply the fix, *save your data*, live happily ever after.

Data classes

Data classes

We talked about *types* so far, next we'll introduce the concept of *classes*.

- "Everything that exists in R is an object". However, not everything is object-oriented (OO).
- The difference between **base** and **OO** objects is that OO objects have a "**class**" attribute.
- The class attribute determines the behavior of an object when passed to a **generic function**.

  You can read more about it here if you're curious.

- Partiremo dai **vettori** e da questi (come se fossero dei Lego) costruiremo altre strutture più articolate.
- Gli esempi che considereremo sono: **factors**, **dates** e **data frames**.

Vectors

- Si distinguono in **atomic vectors** e **lists**.
- Gli elementi degli atomic vectors devono essere tutti dello stesso tipo, mentre possono essere di tipo diverso nelle liste.
- Sono accomunati dal fatto di possedere due proprietà, `typeof()`e `length()`. Eventualmente altri attributi da `attributes()`.
- I quattro principali tipi di vettori sono *logical*, *integer*, *double*, e *character*. Vettori integer e double sono insieme noti come vettori *numeric*. (Ci sono due tipi rari: *complex* e *raw*).
- Per creare un vettore con più di un elemento si usa `c()`, short per *combine*.
- Quando si cerca di combinare tipi differenti questi sono costretti a convertirsi secondo un ordine fisso: character → double → integer → logical.

Esempio

```r
temp <- c(0.2, 0.5, 1.47, 3.82, 0.2, 0.78)
temp
```

```
## [1] 0.20 0.50 1.47 3.82 0.20 0.78
```

```r
# Testing vector type
is.atomic(temp) ; is.list(temp)
```

```
## [1] TRUE
```

```
## [1] FALSE
```

```r
# Testing length and value type
length(temp) ; typeof(temp)
```

```
## [1] 6
```

```
## [1] "double"
```

Attributes

Possiamo costruire strutture più complesse a partire dagli *atomic vectors* aggiungendo degli `attributes`.

- Usando l'attributo `dim` possiamo creare **matrici** (2D) e **arrays** (multidimensionali), mentre l'attributo `class` produce OO objects come **fattori** e **date**.

Matrici

```
# Two scalar arguments specify row and column sizes
x <- matrix(1:6, nrow = 2, ncol = 3)
x
```

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

```
dim(x)
```

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

```
attributes(x)
```

```
## $dim
## [1] 2 3
```

```
# You can also modify an object in place by setting dim()
z <- 1:6
dim(z) <- c(2, 3)
```

Arrays

```r
# One vector argument to describe all dimensions
y <- array(1:12, c(2, 3, 2))
y
```

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

```r
attributes(y)
```

```
## $dim
## [1] 2 3 2
```

Factors

R uses factors to handle **categorical variables**, variables that have a fixed and known set of possible values

```
x <- factor(c("BS", "MS", "PhD", "MS"))
x
```

```
## [1] BS  MS  PhD MS 
## Levels: BS MS PhD
```

```
attributes(x)
```

```
## $levels
## [1] "BS"  "MS"  "PhD"
## 
## $class
## [1] "factor"
```

More on factors

We can think of factors like character (level labels) and an integer (level numbers) glued together

```r
glimpse(x)
```

```
##  Factor w/ 3 levels "BS","MS","PhD": 1 2 3 2
```

```r
as.integer(x)
```

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

I fattori sono costruiti a partire da vettori interi.

Dates

```r
y <- as.Date("2020-01-01")
y
```

```
## [1] "2020-01-01"
```

```r
attributes(y)
```

```
## $class
## [1] "Date"
```

```r
typeof(y)
```

```
## [1] "double"
```

```r
class(y)
```

```
## [1] "Date"
```

More on dates

We can think of dates like an integer (the number of days since the origin, 1 Jan 1970) and an integer (the origin) glued together

```r
as.integer(y)
```

```
## [1] 18262
```

```r
as.integer(y) / 365 # roughly 50 yrs
```

```
## [1] 50.03288
```

Le date sono costruite a partire da vettori numerici.

Lists

Lists are a generic vector container: vectors of any type can go in them

```r
l <- list(
  x = 1:4,
  y = c("hi", "hello", "bye"),
  z = c(TRUE, FALSE)
)
l
```

```
## $x
## [1] 1 2 3 4
##
## $y
## [1] "hi"    "hello" "bye"
##
## $z
## [1]  TRUE FALSE
```

More on lists

```r
length(l)
```

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

```r
typeof(l)
```

```
## [1] "list"
```

```r
attributes(l)
```

```
## $names
## [1] "x" "y" "z"
```

Data frames

We can think of data frames like vectors of equal length glued together.
A data frame is a special list containing vectors (of possible different type) of equal length.

```r
df <- data.frame(x = 1:2, y = c("hi", "hello"))
df
```

```
##   x     y
## 1 1    hi
## 2 2 hello
```

```r
attributes(df)
```

```
## $names
## [1] "x" "y"
##
## $class
## [1] "data.frame"
##
## $row.names
## [1] 1 2
```

The `str` function

```
str(df)
```

```
## 'data.frame':    2 obs. of  2 variables:
##  $ x: int  1 2
##  $ y: chr  "hi" "hello"
```

```
str(l)
```

```
## List of 3
##  $ x: int [1:4] 1 2 3 4
##  $ y: chr [1:3] "hi" "hello" "bye"
##  $ z: logi [1:2] TRUE FALSE
```

```r
str(temp)
```

```
##  num [1:6] 0.2 0.5 1.47 3.82 0.2 0.78
```

```r
str(z)
```

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

```r
str(y)
```

```
##  Date[1:1], format: "2020-01-01"
```

```r
str(x)
```

```
##  Factor w/ 3 levels "BS","MS","PhD": 1 2 3 2
```

Subsetting

```r
temp[2] ; temp[2:3]
```

```
## [1] 0.5
```

```
## [1] 0.50 1.47
```

```r
z[,1] ; z[1,2]
```

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

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

```r
l[1] ; l[[1]] ; l$x
```

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

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

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

```
df
```

```
##   x     y
## 1 1    hi
## 2 2 hello
```

```
df[,1] ; df$x
```

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

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

- We can use the `pull()` function of `dplyr()` pkg to extract a vector from the data frame

```
df %>%
  pull(y)
```

```
## [1] "hi"    "hello"
```