1. Objects and Attributes
2. Matrices
3. Lists
4. Data Frames
5. S3 Objects
6. Importing Data in R
7. Exercises
Intrinsic attributes: mode

All entities in R are called *objects*. Objects have the **intrinsic attributes** *mode* and *length*.

```r
x <- c(1.5, 2.6, 3.7)
x
## [1] 1.5 2.6 3.7
mode(x)
## [1] "numeric"

y <- as.character(x) # coercion with as.<datatype>
y
## [1] "1.5" "2.6" "3.7"
mode(y)
## [1] "character"
```

Modes are types “numeric”, “complex”, “logical”, “character”, and “raw”.

Intrinsic attributes: length

```
x
## [1] 1.5 2.6 3.7
length(x)  # length attribute
## [1] 3
e <- numeric()
e
## numeric(0)
length(e)

## [1] 0
e[5] <- 12  # implicitly changes length
e
## [1] NA NA NA NA 12
length(e) <- 7  # changing the length explicitly
e
## [1] NA NA NA NA 12 NA NA
```
Regular attributes can be read and set using `attr()` and `attributes()`.

```r
z <- 1:4
z
## [1] 1 2 3 4
attributes(z)  # z does not have any attributes
## NULL
length(z)
## [1] 4
class(z)
## [1] "integer"
mode(z)
## [1] "numeric"
storage.mode(z)
## [1] "integer"
```
Setting an attribute can change the object. For example, the dim attribute allows R to treat z as a matrix.

```r
attr(z, "dim") <- c(2,2)
z
## [,1] [,2]  
## [1,] 1 3  
## [2,] 2 4
attributes(z)  # returns attributes as a list
## $dim
## [1] 2 2
length(z)  
## [1] 4
class(z)  # note: the class has changed!
## [1] "matrix"
mode(z)  
## [1] "numeric"
storage.mode(z)  
## [1] "integer"
```
Notes:

- mode, storage.mode, and class are confusing!
- class() returns the class of an object (same as attr(x, 'class')). If the object does not have a class attribute (S3) or an implicit class (matrix, array, integer) then it returns the storage type (storage.mode).

Recommendation

Use ‘str()’ to inspect an object’s structure and attributes instead.

Example:

```r
str(z)
```

```r
## int [1:2, 1:2] 1 2 3 4
```
Section 2

Matrices
Matrix: 2-dimensional array with consistent data type

```r
x <- matrix(1:6, nrow=2, ncol=3)  # create a matrix

x

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

x[2, 2]  # get one element

## [1] 4

x[1,]  # get the first row

## [1] 1 3 5

x[,2]  # get 2nd column

## [1] 3 4

dim(x)  # look at the dimensions

## [1] 2 3

nrow(x)

## [1] 2

ncol(x)

## [1] 3

length(x) # intrinsic length attribute (matrix has 6 values)

## [1] 6
```
Matrix: Dimnames

colnames(x) <- c("X1", "X2", "X3")
x
## X1 X2 X3
## [1,] 1 3 5
## [2,] 2 4 6
rownames(x) <- c("Michael", "peter")
x
## X1 X2 X3
## Michael 1 3 5
## peter 2 4 6
dimnames(x) # names for all dimensions as a list
## [[1]]
## [1] "Michael" "peter"
##
## [[2]]
## [1] "X1" "X2" "X3"
Matrix: Row and Column operations

```
x
##     X1 X2 X3
## Michael 1 3 5
##  peter   2 4 6

rowSums(x)
##     Michael  peter
##          9      12

colSums(x)
##     X1 X2 X3
##            3  7 11

rowMeans(x)
##     Michael  peter
##          3      4

colMeans(x)
##     X1 X2 X3
##       1.5  3.5  5.5

apply(x, MARGIN = 1, FUN = mean)  # same as rowMeans(x)
##     Michael  peter
##          3      4
```

apply applies any function to rows (MARGIN = 1) or columns (MARGIN = 2) of a matrix.
Matrix: `rbind`, `cbind`

Construct a matrix by adding another matrix as new columns or rows.

```r
m1 <- matrix(TRUE, nrow=2, ncol=2)
m0 <- matrix(FALSE, nrow=2, ncol=2)

x <- cbind(m0, m1) # binding columns
x
## [1,] FALSE FALSE TRUE TRUE
## [2,] FALSE FALSE TRUE TRUE

x <- rbind(x, cbind(m1,m0)) # binding rows
x
## [1,] FALSE FALSE TRUE TRUE
## [2,] FALSE FALSE TRUE TRUE
## [3,] TRUE TRUE FALSE FALSE
## [4,] TRUE TRUE FALSE FALSE
```
Matrix algebra (advanced knowledge)

```r
a <- 1:3; b <- 3:1
ab <- outer(a, b, "*") # outer product
ab
## [,1] [,2] [,3]
## [1,] 3 2 1
## [2,] 6 4 2
## [3,] 9 6 3
t(ab) # transpose of a
## [,1] [,2] [,3]
## [1,] 3 6 9
## [2,] 2 4 6
## [3,] 1 2 3
ab*ab # element by element product
## [,1] [,2] [,3]
## [1,] 9 4 1
## [2,] 36 16 4
## [3,] 81 36 9
ab %*% ab # matrix product
## [,1] [,2] [,3]
## [1,] 30 20 10
## [2,] 60 40 20
## [3,] 90 60 30
```

Other important functions: `crossprod()`, `solve()` (linear equations), `svd()`, `eigen()`
Section 3

Lists
Lists are very common in R. A list is an object consisting of an ordered collection of objects (its components).

```r
lst <- list(name = "Fred", wife = "Mary", no.children = 3,
             child.ages = c(4,7,9))

lst
```

```
## $name
## [1] "Fred"
##
## $wife
## [1] "Mary"
##
## $no.children
## [1] 3
##
## $child.ages
## [1] 4 7 9
```

```r
lst[[2]] # access via index
```

```
## [1] "Mary"
```

```r
lst$wife # access via name, also lst[["wife"]]
```

```
## [1] "Mary"
```

```r
str(lst)
```

```
## List of 4
## $ name : chr "Fred"
## $ wife : chr "Mary"
## $ no.children: num 3
## $ child.ages : num [1:3] 4 7 9
```

Lists can contain arbitrary R objects and can be combined with `c()`. Names can be retrieved and changed with `names()`.
Section 4

Data Frames
Data Frame: The *spread sheet* of R

A data frame looks like a spread sheet. It is a list of column vectors with class `data.frame`.

```r
df <- data.frame(name = c("Michael", "Mark", "Maggie"), children = c(2, 0, 2))
df
```

```
## name children
## 1 Michael 2
## 2 Mark 0
## 3 Maggie 2
```

# looks like a list of columns
```r
df$name
```

```
## [1] Michael Mark Maggie
## Levels: Maggie Mark Michael
```

# also looks like a matrix
```r
df[1,]
```

```
## name children
## 1 Michael 2
```

```r
df[, "children"]
```

```
## [1] 2 0 2
```

```r
str(df)
```

```
#' data.frame': 3 obs. of 2 variables:
## $ name : Factor w/ 3 levels "Maggie","Mark",..: 3 2 1
## $ children: num 2 0 2
```

## Hints

1. Data structures can be inspected using the Environment tab in RStudio.
2. A data frame is a list of columns and can be accessed like a list.
3. Character strings are often automatically converted to `factor`. 

---

Michael Hahsler (OIT, SMU)  Intro to R - 2. Objects and Data 18 / 31
Section 5

S3 Objects
S3 objects are just regular R objects with a \texttt{class} attribute. Often it is a list.

```r
# roll a die 100 times and tabulate the results
dice_rolls <- sample(1:6, size = 100, replace = \texttt{TRUE})
tbl <- table(dice_rolls)
tbl
```
```
## dice_rolls
## 1  2  3  4  5  6
## 15 14 14 18 14 25
```

```
attributes(tbl)
```
```
## $dim
## [1] 6
##
## $dimnames
## $dimnames$dice_rolls
## [1] "1" "2" "3" "4" "5" "6"
##
## $class
## [1] "table"
```

\texttt{str()} is very helpful and shows the class.

```r
str(tbl)
```
```
## 'table' int [1:6(1d)] 15 14 14 18 14 25
## - attr(*, "dimnames")=List of 1
## ..$ dice_rolls: chr [1:6] "1" "2" "3" "4" ...
```
Section 6

Importing Data in R
Accessing R data sets

R (or packages) come with data sets. These sets can be loaded using `data()`. Without arguments `data()` shows all available data sets.

```r
data(iris)  # load the iris data set. try: ?iris
ehead(iris) # head shows the first few elements
```

```r
## 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
```

```r
str(iris)  # displays the structure of an object
```

```r
## 'data.frame': 150 obs. of 5 variables:
##  $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...  
##  $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...  
##  $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...  
##  $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...  
##  $ Species : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```r
summary(iris)
```

```r
## Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## Min. :4.3 Min. :2.0 Min. :1.0 Min. :0.1    setosa :50
## 1st Qu.:5.1 1st Qu.:2.8 1st Qu.:1.6 1st Qu.:0.3 versicolor:50
## Median :5.8 Median :3.0 Median :4.3 Median :1.3  virginica :50
## Mean :5.8 Mean :3.1 Mean :4.3 Mean :1.3
## 3rd Qu.:6.4 3rd Qu.:3.3 3rd Qu.:5.1 3rd Qu.:1.8
## Max. :7.9 Max. :4.4 Max. :6.9 Max. :2.5
```
**Reading and writing CSV and text files**

`read.table()` and `write.table()` can be used to read/write complete file to/from data.frames. The file format can be space or tab-separated, CSV, with or without column/row labels, etc. For CSV we have a convenience function `read.csv()` and `write.csv()`

```r
df
## name children
## 1 Michael 2
## 2  Mark  0
## 3  Maggie 2

write.csv(df, file="df.csv")
df2 <- read.csv("df.csv")
df2
## X name children
## 1 1 Michael 2
## 2 2  Mark  0
## 3 3  Maggie 2

unlink("df.csv")  # remove the file
```

**Notes**

1. Don't forget to set the working directory (RStudio: Session -> Set Working Directory)
2. See `? read.table` and `? write.table` for all available options (column headers, etc.).
# Importing Excel Files

**Note:** You need to

- Install package `xlsx` using `Tools -> Install Packages...`
- download the file `MLB_cleaned.xlsx`

```r
library(xlsx)
mlb <- xlsx::read.xlsx2("MLB_cleaned.xlsx", sheetIndex = 1)

Important: Always check if the data was read in correctly!!!

```
```
```
```
Importing Files: Common issues

Data looks fine with head(), but...

```r
str(mlb)
```

## 'data.frame': 1034 obs. of 7 variables:
## $ First.Name : Factor w/ 435 levels "A.J.","Aaron",...: 209 293 235 182 247 65 344 376 389 333 ...
## $ Last.Name : Factor w/ 849 levels "Aardsma","Abercrombie",...: 488 551 525 401 530 417 626 356 241 828 ...
## $ Team : Factor w/ 30 levels "ANA","ARZ","ATL",...: 1 1 1 1 1 1 1 1 1 1 ...
## $ Position : Factor w/ 9 levels "Catcher","Designated Hitter",...: 1 1 1 3 3 3 3 3 4 4 ...
## $ Height.inches.: Factor w/ 17 levels "67","68","69",...: 6 6 8 4 7 9 7 7 9 5 ...
## $ Weight.pounds.: Factor w/ 89 levels "150","155","156",...: 16 41 56 16 56 46 36 47 36 21 ...
## $ Age : Factor w/ 725 levels "20.9","21.46",...: 78 164 534 64 67 84 442 528 148 193 ...

**Recommendation**

- Often data types are not read correctly (e.g., even numbers might be read as a factor) and variable names have extra dots at the end. These need to be fixed (see next slide).
- There may be issues with different versions of Excel. It is often easier to export the spreadsheet from Excel as a CSV file to be read by R.
Importing Files: Fixing data types

```r
mlb$Height <- as.numeric(as.character(mlb$Height.inches.))
mlb$Height.inches. <- NULL # remove the column with the old name
mlb$Weight <- as.numeric(as.character(mlb$Weight.pounds.))
mlb$Weight.pounds. <- NULL
mlb$Age <- as.numeric(as.character(mlb$Age))

str(mlb)
```

```
## 'data.frame': 1034 obs. of 7 variables:
## $ First.Name: Factor w/ 435 levels "A.J." , "Aaron", ... : 209 293 235 182 247 65 344 376 389 333 ...
## $ Last.Name : Factor w/ 849 levels "Aardsma", "Abercrombie", ... : 488 551 525 401 530 417 626 356 241 828 ...
## $ Team     : Factor w/ 30 levels "ANA", "ARZ", "ATL", ... : 1 1 1 1 1 1 1 1 1 1 ... 
## $ Position : Factor w/ 9 levels "Catcher", "Designated Hitter", ... : 1 1 1 3 3 3 3 3 4 4 ...
## $ Age      : num 23.9 25.3 31.7 23.6 23.7 ... 
## $ Height   : num 72 72 74 70 73 75 73 75 71 ... 
## $ Weight   : num 180 205 220 180 220 210 200 211 200 185 ... 
```

This looks better. Height, weight and age are now numbers.
Importing Files: Fixing data types

```r
summary(mlb)
```

```r
## First.Name Last.Name Team Position
## Jason : 27 Johnson : 9 NYM : 38 Relief Pitcher :315
## Chris : 26 Perez : 7 ATL : 37 Starting Pitcher:221
## Mike : 26 Gonzalez : 6 DET : 37 Outfielder :194
## Scott : 24 Hernandez: 6 OAK : 37 Catcher : 76
## Ryan : 23 Jones : 6 BOS : 36 Second Baseman : 58
## Matt : 19 Ramirez : 6 CHC : 36 First Baseman : 55
## (Other):889 (Other) :994 (Other):813 (Other) :115

## Age Height Weight
## Min. :21 Min. :67 Min. :150
## 1st Qu.:25 1st Qu.:72 1st Qu.:187
## Median :28 Median :74 Median :200
## Mean :29 Mean :74 Mean :202
## 3rd Qu.:31 3rd Qu.:75 3rd Qu.:215
## Max. :49 Max. :83 Max. :290

mlb_avg <- aggregate(mlb[, c("Age", "Height", "Weight")],
                      by = list(Team = mlb$Team), FUN = mean)
head(mlb_avg)
```

```r
## Team Age Height Weight
## 1 ANA 29 73 201
## 2 ARZ 28 74 208
## 3 ATL 28 74 200
## 4 BAL 29 73 196
## 5 BOS 30 74 205
## 6 CHC 28 74 204
```
Exporting Data to Excel

Options:

1. Write data into a CVS file with `write.csv()` and open in Excel.

2. Construct a Excel workbook using multiple calls to `write.xlsx2()`

```r
library("xlsx")

# create a workbook with two spread sheets
write.xlsx2(mlb, file = "mlb_new.xlsx",
            sheetName = "MLB Data")
write.xlsx2(mlb_avg, file = "mlb_new.xlsx",
            sheetName = "MLB Averages", append = TRUE)
```
Importing an Excel sheet that was saved as CSV.

It is often better to just save the Excel sheet as a CSV file and read it into R.

```r
mlb <- read.csv(paste0("https://michael.hahsler.net/SMU/", 
  "DS_Workshop_Intro_R/examples/MLB_cleaned.csv"))
# paste0 is just used because the URL is too long to fit on the slide
str(mlb)
```

```r
## 'data.frame': 1034 obs. of 7 variables:
## $ First.Name  : Factor w/ 435 levels "A.J.","Aaron",..: 209 293 235 182 247 65 344 376 389 333 ...
## $ Last.Name   : Factor w/ 849 levels "Aardsma","Abercrombie",..: 488 551 525 401 530 417 626 356 241 828 ...
## $ Team        : Factor w/ 30 levels "ANA","ARZ","ATL",..: 1 1 1 1 1 1 1 1 1 1 ...
## $ Position    : Factor w/ 9 levels "Catcher","Designated Hitter",..: 1 1 1 1 1 1 1 1 1 1 ...
## $ Height.inches: int 72 72 74 70 73 75 73 73 75 71 ...
## $ Weight.pounds: int 180 205 220 180 220 210 200 211 200 185 ...
## $ Age         : num 23.9 25.3 31.7 23.6 23.7 ...
```

**Hint**

R can directly read from URLs.
Section 7

Exercises
Exercises

1. Read and clean the MLB data set.
2. Select only the players for the team ‘ARZ’. Compare the column Team with ‘ARZ’ and use subsetting to select the rows.
3. How many players does the team ‘ARZ’ have in the data set?
4. What is the weight of the heaviest player of the team ‘ARZ’ (use a function).
5. What is the average age of all players in the dataset?
6. Add a column called BMI and add the body mass index (https://en.wikipedia.org/wiki/Body_mass_index) for each player.
7. Create a data.frame containing the names, year of birth, month of birth and day of birth as separate columns with the information for 3 people. Make sure the data.frame has column names (see colnames()).
8. Write the data.frame to a file in CSV format and check it in Excel.