Advanced R Programming - Lecture 4

Krzysztof Bartoszek
(slides based on Leif Jonsson's and Måns Magnusson's)

Linköping University

krzysztof.bartoszek@liu.se

10 September 2019 (C2)

Today

Linear algebra using R

Dynamic reporting with knitr and R-markdown

ggplot2

Object orientation

Questions since last time?

Big Bang Theory!

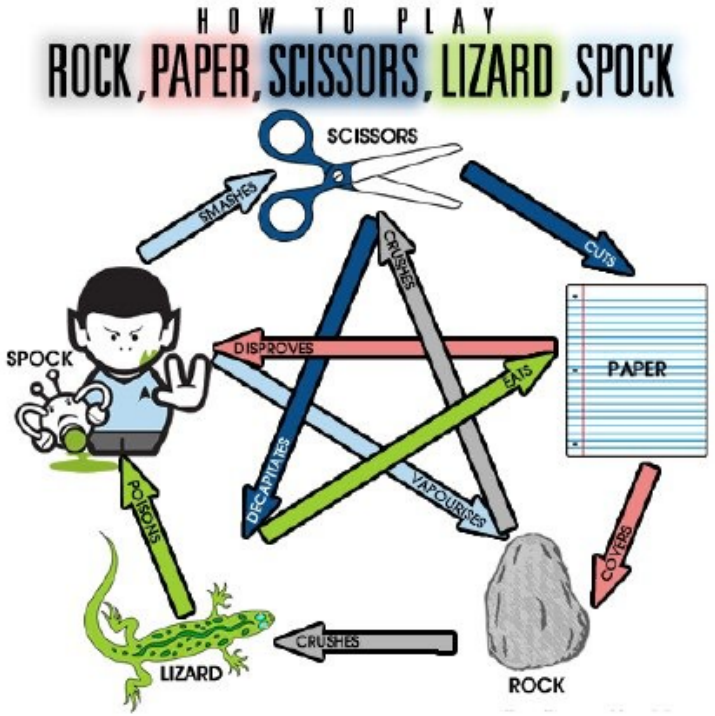


Figure: Rock-paper-scissors according to Sheldon!

http://www.fanpop.com/clubs/the-big-bang-theory/images/34015104/title/rock-paper-scissors-lizard-spock-fanart

sheldon_game

```
sheldon_game <- function(player1, player2){
  alt <- c("rock", "lizard", "spock", "scissors", "paper")
  stopifnot(player1 %in% alt, player2 %in% alt)
  alt1 <- which(alt %in% player1)
  alt2 <- which(alt %in% player2)

  if(any((alt1 + c(1,3)) %% 5 == alt2)) {
        return("Player 1 wins!")
  } else {
        return("Player 2 wins!")
  }
  return("Draw!")
}
```

Linear algebra in R

Basics in base

Uses LINPACK or LAPACK

Extra functionality : Matrix package
(extra LAPACK functionality)

Linear algebra

```
# Create matrix
A <- matrix(1:9,ncol=3)

# Block matrices
cbind(A,A); rbind(A,A)

# Transpose
t(A)

# Addition and subtraction
A + A; A - A

# Matrix multiplication
A%*%A

# Matrix inversion
solve(A)
```

Linear algebra

```
# Eigenvalues
eigen(A)

# Determinants
det(A)

# Matrix factorization
svd(A)
qr(A)

# Cholesky decomposition
chol(A)
```

Donald E. Knuth, Literate Programming, 1984

Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to humans what we want the computer to do.
- Donald E. Knuth, Literate Programming, 1984

Background

Reproducible research

Literate programming

Dynamic (repeated) reports

(Tutorials)

markdown

simple markup language

alternative to HTML (and LaTeX)

developed further by R-studio
(see coursepage)

knitr + md = rmd

Add R to markdown

knitr + md = rmd

Add R to markdown

(a)
.rmd

Figure: Flow

knitr + md = rmd

Add R to markdown

(a)
.rmd

(b) knitr

Figure: Flow

knitr + md = rmd

Add R to markdown

(a) .rmd

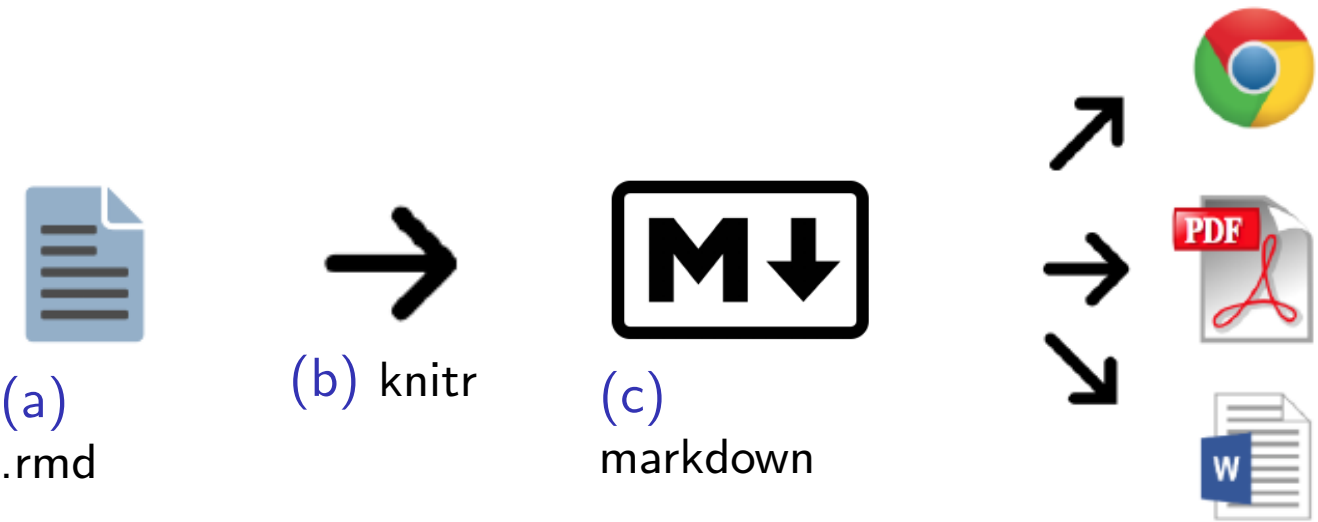
(b) knitr

(c) markdown

Figure: Flow

knitr + md = rmd

Add R to markdown



(a) .rmd

(b) knitr

(c) markdown

(d) pandoc

Figure: Flow

ggplot2

popular visualization package

"The grammar of graphics"
- the language of visualization

flexible

ggplot examples:
`http://shiny.stat.ubc.ca/r-graph-catalog/`

the grammar

Create a graph layer by layer

Store as object (print to plot)

Three (main) parts:

`data`	The data to visualize (data.frame)
`geom`	The geometric representation of data
`aes`	The mapping of colors/shape to data

geom

`geom_point`	Scatterplots
`geom_line`	Lineplots
`geom_boxplot`	Boxplot
`geom_histogram`	Histograms
`geom_bar`	Barchart

aes

`x`

`y`

`size`

`color`

`shape`

Special aes

geom	Special aes
geom_point	point shape, point size
geom_line	line type, line size
geom_bar	y min, y max, fill color, outline color

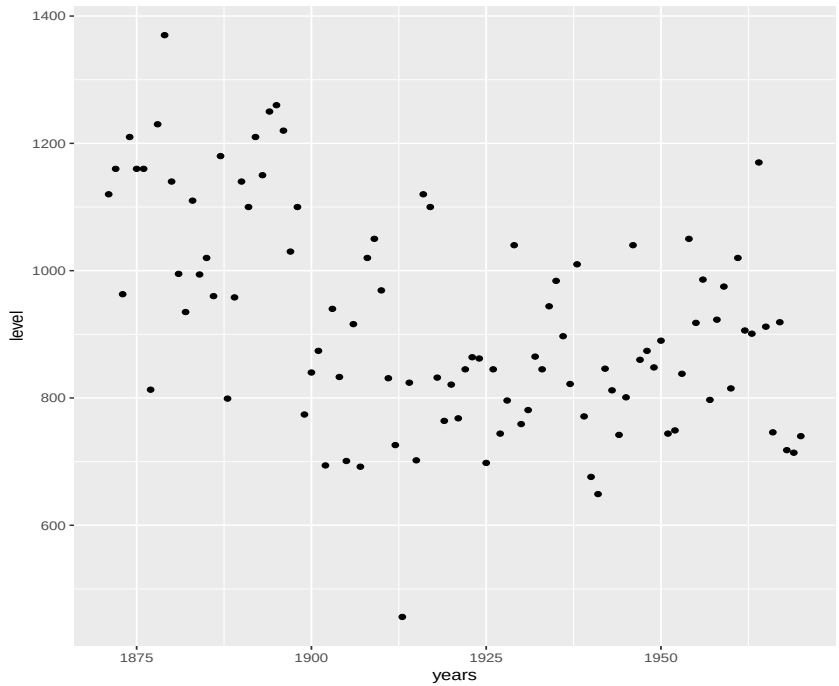
GGPlot2: Example

```
library(ggplot2)

# Preprocessing
data(Nile)
Nile <- as.data.frame(Nile)
colnames(Nile) <- "level"
Nile$years <- 1871:1970
Nile$period <- "- 1900"
Nile$period[Nile$years>=1900]<-"1900 - 1945"
Nile$period[Nile$years > 1945] <- "1945 +"
Nile$period <- as.factor(Nile$period)
```

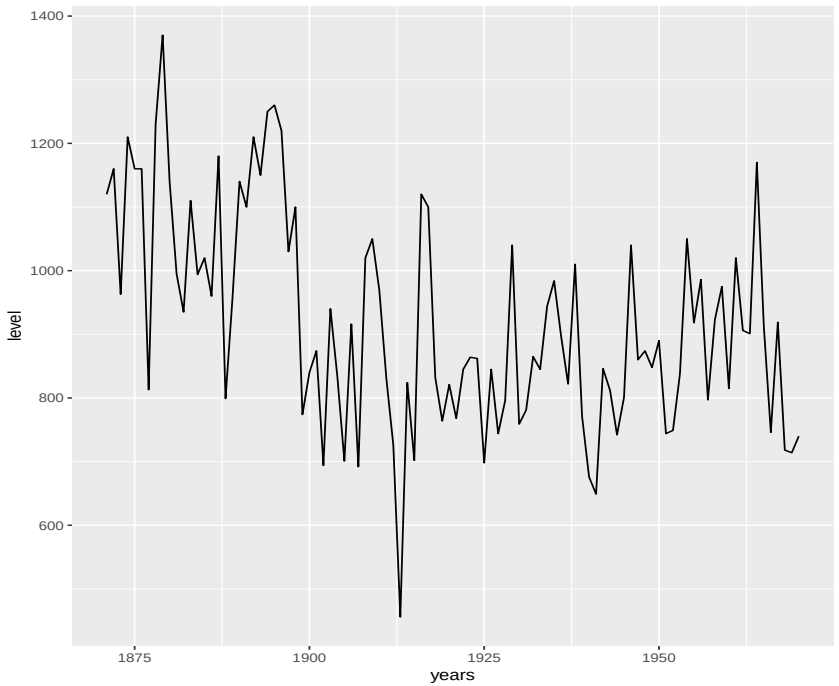
GGPlot2: geom_point

```
pl <-
        ggplot(data=Nile) +
        aes(x=years, y=level) +
        geom_point()
pl
```



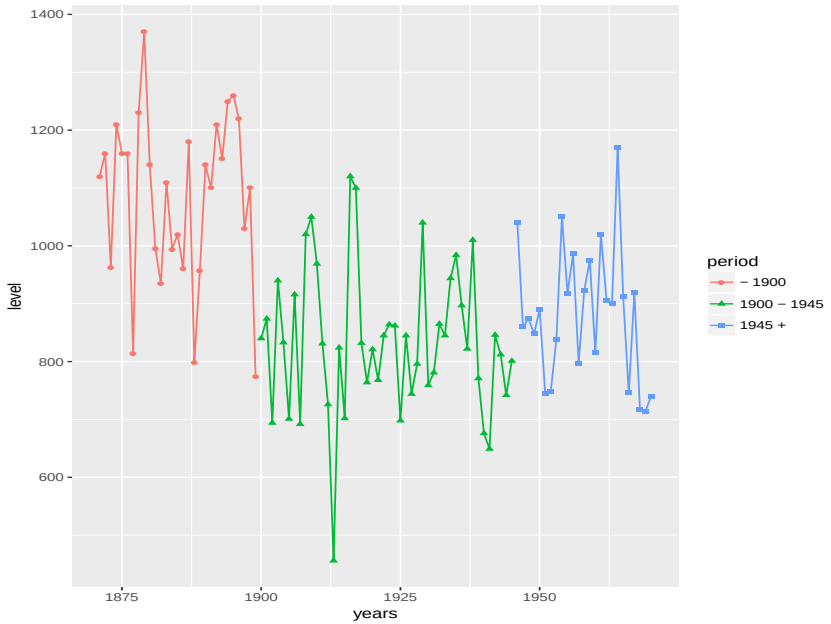
GGPlot2: geom_line

```
pl <-
        ggplot(data=Nile) +
        aes(x=years, y=level) +
        geom_line()
pl
```



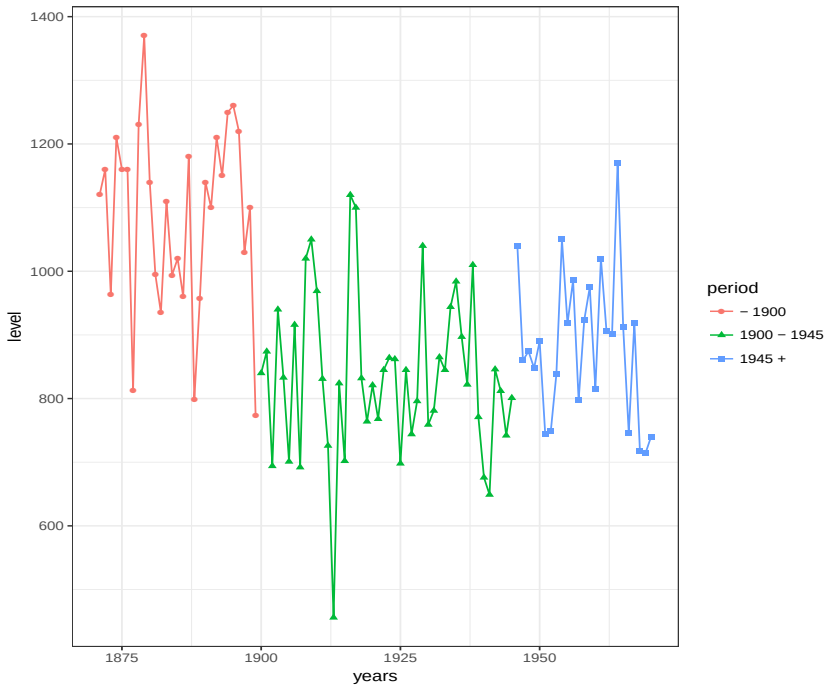
GGPlot2: geom_point + geom_line + colors!

```
pl <-
        ggplot(data=Nile) +
        aes(x=years, y=level, color=period) +
        geom_line(aes(type=period)) +
        geom_point(aes(shape=period))
pl
```



GGPlot2: use BW theme

```
pl + theme_bw()
```



Object orientation

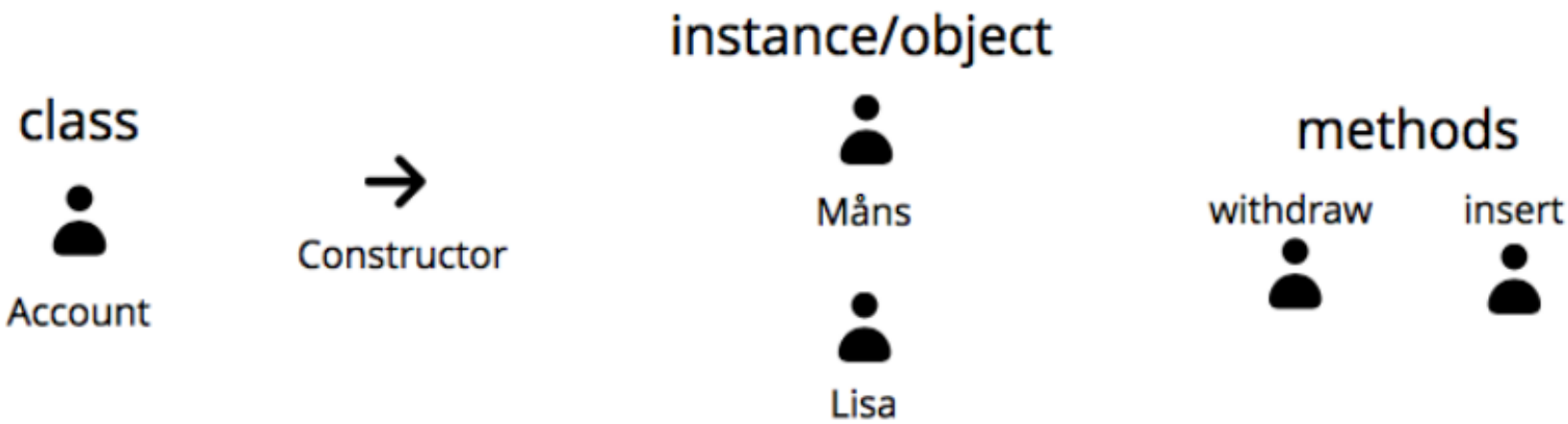
Programming paradigm

Mutable states

Key abstraction is "an object"

R is *not* purely object oriented

Object orientation



Object orientation

Fields

currency (12/24) : class variable

current_amount : object variable

no_withdraws : object variable

Methods

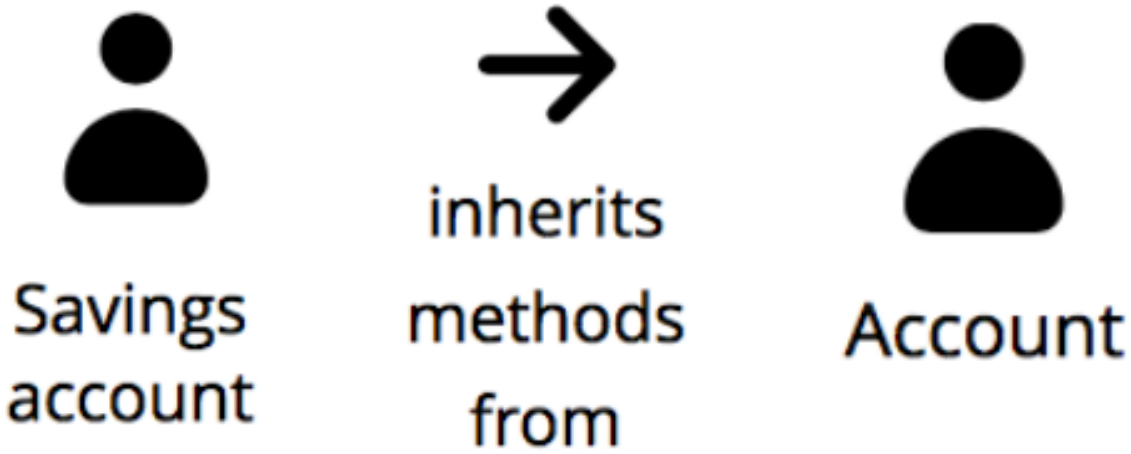
insert()

withdraw()

Object orientation

- **class**: template and generator of objects
- **constructor**: minimal set all fields to default values (organize the memory allocated for the object)
- **interface**: collection of "services" the class/object offers, *public interface*, need to be exported in `NAMESPACE`

Inheritance



Possible from multiple classes, your class can extend more than one class.

Object orientation in R

S3
Simple
Methods belongs to functions

Object orientation in R

S3	S4
Simple	More formal
Methods belongs to functions	Methods belongs to functions
	@Fields
	Parents

Object orientation in R

S3	S4	RC
Simple	More formal	Latest (R 2.12)
Methods belongs to functions	Methods belongs to functions	no copy-on-modify
	@Fields	Methods belongs to objects
	Parents	Objects have Fields and methods $

S3

- "Not really objects, more of a naming convention"
- "Based around the `.` syntax: E.g. for `print`, `print` calls `print.lm` `print.anova`, etc. And if not found, `print.default`"

https://stackoverflow.com/questions/9521651/r-and-object-oriented-programming

```
# Create object
x <- 1:100
class(x) <- "my_numeric"
```

S3

Methods belong to functions (the generic ones)

```
# Create object
x <- 1:100
class(x) <- "my_numeric"

# Create generic function
# S3 classes have own implementation
# of a function called f
f <- function(x) UseMethod("f")
```

S3

`print()` is a generic function

```
# Create object
x <- 1:100
class(x) <- "my_numeric"

# Create generic function
# S3 classes have own implementation
# of a function called f
f <- function(x) UseMethod("f")

# Create method
print.my_numeric <- function(x, ...){
        cat("This is my numeric vector.")
}
```

call: `print(x)`

S3

```
# Create object
x <- 1:100
class(x) <- "my_numeric"

# Create generic function
# S3 classes have own implementation
# of a function called f
f <- function(x) UseMethod("f")

# Create method
print.my_numeric <- function(x, ...){
        cat("This is my numeric vector.")
}
```

Usage of `.` discouraged in names of own functions and objects.
`t.test()`: `t` method for `test` objects?
(typo p.103 of printed book, Ed. 1, online is correct)

S4

```
# Create class with slots (with permitted classes)
setClass("Person",
slots=list(name="character", age="numeric",
 salary="numeric"))
# Create inheriting class, can inherit from multiple
setClass("Employee",
slots=list(boss="Person"),contains="Person")

alice<-new("Person", name="Alice",age=40, salary=100)
alice@age
bob<-new("Employee", name="Bob",age=25, salary=100,
boss=alice)
```

S4: Methods: create a generic, then instances

```
setGeneric("salary_change", function(p, i) {
   standardGeneric("salary_change")
 })
setMethod("salary_change",
signature(p = "Person", i = "numeric"),
  function(p, i) {
    p@salary+i
  })
setMethod("salary_change",
signature(p = "Employee", i = "numeric"),
  function(p, i) {
     nsal<-callNextMethod()
     ## method from parent (contained) class
     if (nsal>p@boss@salary){nsal<-p@salary}
     nsal
  })
```

RC

```
# Create object with fields and methods
Account <- setRefClass("Account",
        fields = list(balance = "numeric"),
        methods = list(
                withdraw = function(x) {
                        balance <<- balance - x
                },
                deposit = function(x) {
                        balance <<- balance + x
                }
        )
)
```

RC: objects are mutable

```
a<-Account$new(balance=100)

a$balance<-200; a$balance ##output: 200
b<-a;b$balance ##output: 200
a$balance<-0;b$balance ##output: 0

c<-a$copy() ## all RC objects have a copy() method

a$balance<-100;c$balance;a$balance ##output: 0, 100

## S4: if we change something in alice,
## then bob's boss does not change
salary_change(bob,5) ##output: 100
alice@salary<-salary_change(alice,10)
salary_change(bob,5) ##output: 100
bob@boss<-alice; salary_change(bob,5) ##output: 105
```

NAMESPACE: exporting **(LABS!)**

S3

`S3method(method, class)`

e.g. `S3method(print, my_numeric)`

S4

in `DESCRIPTION Depends: methods`

`exportClasses(class)`: class publicly avaialable,

`export(class)`: generator publicly avaialable

`http://stat.ethz.ch/R-manual/R-devel/library/methods/html/Introduction.html`

`exportMethods(method)`: "If a package defines methods for generic functions, those methods should be exported if any of the classes involved are exported", e.g. `plot()`

`https://stat.ethz.ch/R-manual/R-patched/library/methods/html/setMethod.html`

RC: same as S4 but typically impossible to be extended outside your package

NAMESPACE: importing **(LABS!)**

S4

```
importClassesFrom(package, ...)
importMethodsFrom(package, ...)
```

http://www.hep.by/gnu/r-patched/r-exts/R-exts_33.html

More OO classes

`https://stackoverflow.com/questions/9521651/r-and-object-oriented-programming`

- Reference classes `?setRefClass`, "Primarily useful to avoid making copies of large objects (pass by reference)"
- proto "Neat concept (prototypes, not classes), but seems tricky in practice"
- R6 ""'Creating an R6 class is similar to the reference class, except that theres no need to separate the fields and methods, and you cant specify the types of the fields.'""
- R.oo

The End... for today.
Questions?
See you next time!