# Advanced R Programming - Lecture 1

Leif Jonsson

Linköping University

*leif.jonsson@ericsson.com*
*leif.r.jonsson@liu.se*

August 16, 2016

# Today

# Learn to

- Write R programs and packages
- Write performant code
- Learn basic software engineering practices
- Solve (machine learning) problems using R

# But most important...

# But most important...

Your primary tool in the next 2 years

# Course Plan

## Part 1: R Syntax

Period: Week 1-2
Students work: Individually
Lab: Documented R file
Computer lab

## Topics

- Basic R Syntax
- Basic data structures
- Program control
- R packages

## Part 2: Advanced topics

Period: Week 3-7
Students work: In groups
Turn in: R package on GitHub
Seminar

## Topics

- Performant code: Writing quality code
- Linear algebra, Object orientation, Graphics
- Advanced I/O
- Performant code: Writing fast code
- Machine learning in R

## Today

# Presentation(s)

# Me - AKA, Leif Jonsson

**My background**

1. Computer Science, Uppsala 1998
2. Ericsson
3. PhD Applied Machine Learning, LiU, PELAB - STIMA


Figure: Me

# You

- Backgound?
- Why this course?
- Expectations?

# Course Practicals...

# Course Practicals...

- Course code: 732A94
- https://www.ida.liu.se/~732A94/index.en.shtml
- https://github.com/MansMeg/AdvRCourse
- https://www.rstudio.com/
- https://cran.r-project.org/
- https://git-scm.com/

# Course litterature...

# Course litterature...

- Matloff, N. The art of R programming [online]
- Wickham, H. Advanced R [online]
- Wickham, H. R packages [online]
- ...and articles.

# Examination

Weekly mandatory labs/projects - deadline Wednesdays 18.00
Computer exam

# Why R?

# Pros

- Popular (among statisticians)
- Open source - all major platforms!
- High-level language - focus on data analysis
- Strong community - vast amount of packages
- Powerful for communicating results
- Connections to high-performance languages as C/C++ and Java

# Cons

- "Ad hoc" language (Compare Perl, Awk, Sh...)
- Can be slooooow
- Can be memory inefficient
- (Still) Hard to troubleshoot
- (Still) Inferior IDE support compared to state of the art

# Pros/Cons

- Niche language
- Specialized syntax
- Very permissive

# Variable types

| Variable type | Short | typeof() | R example |
|---|---|---|---|
| Boolean | logi | logical | TRUE |
| Integer | int | integer | 1L |
| Real | num | double | 1.2 |
| Complex | cplx | complex | 0+1i |
| Character | chr | character | "I <3 R" |

# Variable types

| | Variable type | Short | typeof() | R example | |
|---|---|---|---|---|---|
| ⇓ | Boolean | logi | logical | TRUE | ⇓ |
| | Integer | int | integer | 1L | |
| Coersion | Real | num | double | 1.2 | Coersion |
| | Complex | cplx | complex | 0+1i | |
| ⇓ | Character | chr | character | "I <3 R" | ⇓ |

# Data structures

| Dimension | Homogeneous data | Heterogeneous data |
|---|---|---|
| 1 | vector | list |
| 2 | matrix | data.frame |
| n | array | |

- Constructors: vector() list() ...
- Name dimensions: dimnames()

# Arithmetics

- Vectorized operations (element wise)
- Recycling
- Statistical functions

See reference card...

# Logic operators

| In symbols | A | B | $\neg$A | A$\wedge$B | A$\vee$B |
|---|---|---|---|---|---|
| In R | $A$ | $B$ | !$A$ | A&B | $A \vert B$ |
| | TRUE | FALSE | ? | ? | ? |
| | TRUE | TRUE | ? | ? | ? |
| | FALSE | FALSE | ? | ? | ? |
| | FALSE | TRUE | ? | ? | ? |

# Logic operators

| In symbols | A | B | ¬A | A∧B | A∨B |
|---|---|---|---|---|---|
| In R | *A* | *B* | !*A* | A&B | *A*\|*B* |
| | TRUE | FALSE | FALSE | ? | ? |
| | TRUE | TRUE | ? | ? | ? |
| | FALSE | FALSE | ? | ? | ? |
| | FALSE | TRUE | ? | ? | ? |

# Logic operators

| In symbols | A | B | ¬A | A∧B | A∨B |
|---|---|---|---|---|---|
| In R | $A$ | $B$ | $!A$ | A&B | $A \vert B$ |
| | TRUE | FALSE | FALSE | FALSE | ? |
| | TRUE | TRUE | ? | ? | ? |
| | FALSE | FALSE | ? | ? | ? |
| | FALSE | TRUE | ? | ? | ? |

# Logic operators

| In symbols | A | B | $\neg$A | A$\wedge$B | A$\vee$B |
|---|---|---|---|---|---|
| In R | $A$ | $B$ | !$A$ | A&B | $A \vert B$ |
| | TRUE | FALSE | FALSE | FALSE | TRUE |
| | TRUE | TRUE | ? | ? | ? |
| | FALSE | FALSE | ? | ? | ? |
| | FALSE | TRUE | ? | ? | ? |

# Logic operators

| In symbols | A | B | ¬A | A∧B | A∨B |
|---|---|---|---|---|---|
| In R | $A$ | $B$ | $!A$ | A&B | $A \vert B$ |
| | TRUE | FALSE | FALSE | FALSE | TRUE |
| | TRUE | TRUE | FALSE | TRUE | TRUE |
| | FALSE | FALSE | TRUE | FALSE | FALSE |
| | FALSE | TRUE | TRUE | FALSE | TRUE |

# Logic operators

| | | | |
|---|---|---|---|
| In symbols | $\wedge_{i=1}^{N} a_i$ | $\vee_{i=1}^{N} a_i$ | $\{j : a_j == TRUE\}$ |
| In R | $all(A)$ | $any(A)$ | $which(A)$ |

# Relational operators

| | | | | | |
|---|---|---|---|---|---|
| In symbols | $a<b$ | $a \leq b$ | $a \neq b$ | $a = b$ | $a \in b$ |
| In R | $a < b$ | $a <= b$ | $a! = b$ | a == b | a %in% b |

# Vectors

- Use []
- index by:
  - positive integers: include element(s)
  - negative integers: exclude element(s)
  - logical: include TRUEs

```
vect <- c(6,7,8,9)
> vect[vect>7]
[1] 8 9
> vect[1:2]
[1] 6 7
> vect[c(1,2)]
[1] 6 7
> vect[c(-1,-2)]
[1] 8 9
```

# Matrices

- Use [,]
- Two dimensions
- Index as vectors
- Can reduce (drop class) to vector

# Matrices

```
> mat <- matrix(c(1,2,3,4,5,6),nrow=2)
> mat
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> mat[c(1,2),c(1,2)]
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> mat[c(1,2),]
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> mat[mat>4]
[1] 5 6
```

# Lists

- Use [] to access list elements
- Use [[]] to access list content
- Index as vectors
- Use $ to access list element by name
- Not like typical lists in other programming languages

# Lists

```
> lst <- list(a=47,b=11)
> lst[1]
$a
[1] 47

> lst[[1]]
[1] 47
> lst$b
[1] 11
```

# Data frames

- Very powerful data structure
- Can roughly think about it as the R representation of a CSV file
- Can be loaded from a CSV file
- Can be accessed both as a matrix and a list

# Assigning subsets

- Change values in data structures
- Works for all above mentioned data types

# Assigning subsets

```
> mat
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> mat[mat>4] = 75
> mat
     [,1] [,2] [,3]
[1,]    1    3   75
[2,]    2    4   75
```

# Functions

```
my_function_name <- function(x, y){
        z <- x^2 + y^2
        return(z)
}
```

Unlike in many languages `return` in R is a **function**. In other languages, `return` is usually a **reserved word** (like `if`). This means you must use `return` as a function call with parenthesis. By default R returns the last computed value of the function, so return is not strictly necessary in simple cases.

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