Advanced R Programming - Lecture 2

Krzysztof Bartoszek
(slides by Leif Jonsson and Måns Magnusson)

Linköping University

krzysztof.bartoszek@liu.se

30 August 2019 (A1)

Today

- Program Control
- Functions
- Environments and scoping
- Function arguments
- Returning values
- Specials
- Functionals
- Functional programming
- R packages

Questions since last time?

Program Control

Two main components

- Conditional statements
- Loops

See also extra video on program control on course page

Conditional statements

```
if(boolean expression) {
# statements
} else if (boolean expression) {
# statements
} else {
# statements
}
```

Brackets "{...}":

- not needed if single line follows `if`, `else if`, `else`
- but defensive programming

Loops

- for
- while
- repeat

Brackets "{...}":

- again not needed if single line follows `for`, `while`
- but defensive programming

See also extra video on program control on course page

For loop

```
for (name in vector){
# statements
}
```

While loop

```
while (boolean expression){
# statements
}
```

Controlling loops

- break (loop)
- next (iteration)

Repeat loop

```
repeat {
# statements
}
```

- `repeat` needs `break` statement
- brackets "{...}" needed
- unless empty loop: `repeat break`

Functions revisited

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

Function components

Function arguments
Function body
Function environment

These can be accessed in R by:
formals(f)
body(f)
environment(f)

Lexical scoping

(or how does R find stuff?)

Current environment $\Rightarrow$
Parent environment $\Rightarrow$
...
Global environment $\Rightarrow$
... along searchpath to...
Empty environment (fail)

Environment search path

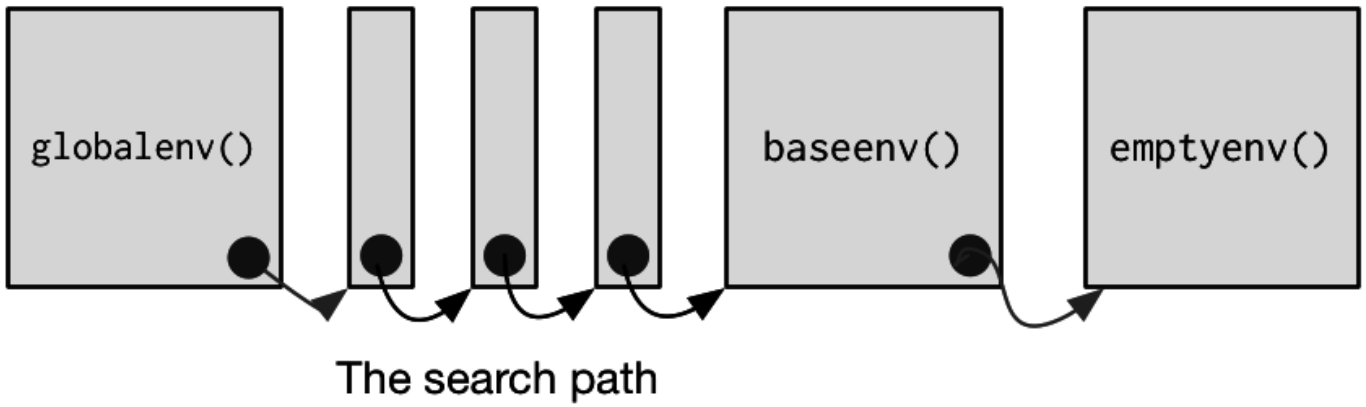

Figure: Environment search–path (H. Wickham, Adv. R, p.127)

```
parent.of.global<-parent.env(.GlobalEnv)
grandparent.of.global<-parent.env(parent.of.global)
```

Environment basics

"bag of names"

```
e <- new.env()
e$a <- FALSE
e$b <- "a"
e$c <- 2.3
e$d <- 1:3
```

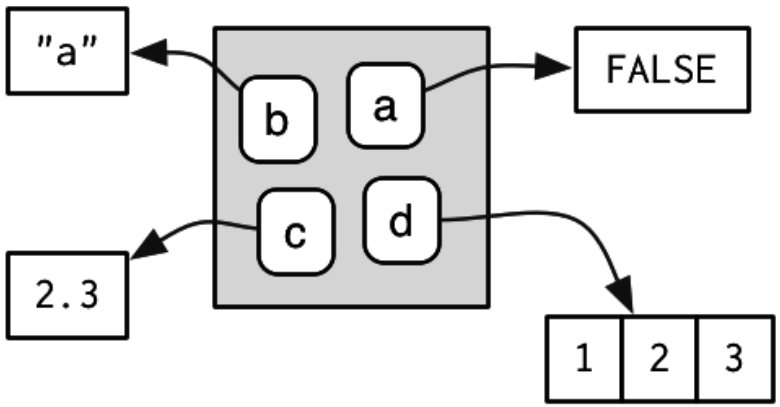

Figure: Environment (H. Wickham, Adv. R, p.125)

Environment relatives

Parents, but no children

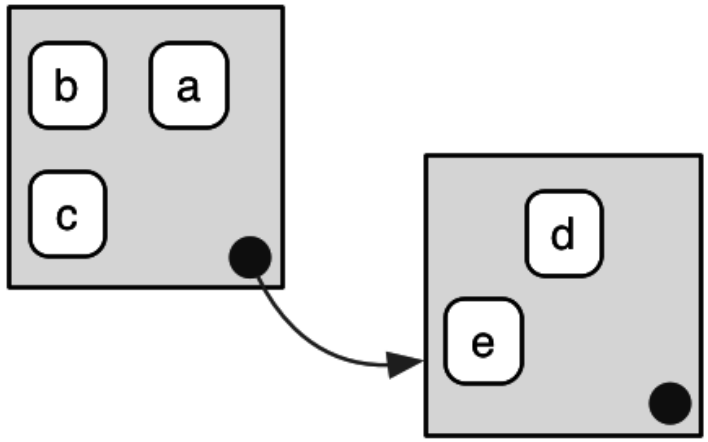

Figure: Env. relations (H. Wickham, Adv. R, p.126)

Working with environments

See environments as lists "of stuff in the bag"

```
ls()
```

Assignments

Shallow assignment
(inside current environment)
<−

Deep assignment
(inside parental environment, if not found, then assign in global)
<<−

Full control assignment
(manually specify environment)
`assign()`

Function arguments

copy-on-modify semantics

"modifying a function argument does not change the original value"

specify arguments by...

position
complete name
partial name

```
myfun(1,2)
myfun(firstarg=1,secondarg=2)
myfun(f=1,s=2)
```

Just in case: partial names cannot be used in function body

Function arguments (cont)

copy-on-modify semantics

`do.call()`
`missing()`
...
Default values

```
do.call(myfun,list(1,2))
do.call("myfun",list(f=1,s=2))
```

`missing()`: check if argument passed

Return values: the last expression evaluated in a function

Multiple values using lists

Pure functions "map the same input to the same output and have no other impact on the workspace. In other words, pure functions have no side effects: they don't affect the state of the world in any way apart from the value they return." (H. Wickham, Adv. R, p.94)

Return values: the last expression evaluated in a function

Multiple values using lists

Pure functions "map the same input to the same output and have no other impact on the workspace. In other words, pure functions have no side effects: they don't affect the state of the world in any way apart from the value they return." (H. Wickham, Adv. R, p.94)

on.exit()

"gets called when the function exits, regardless of whether or not an error was thrown. This means that its main use is for cleaning up after risky behaviour." `https://stackoverflow.com/questions/28300713/how-and-when-should-i-use-on-exit`

return()

Specials

"Most functions in R are prefix operators: the name of the function comes before the arguments." (H. Wickham, Adv. R)

infix functions

"function name comes in between its arguments, like + or -" (H. Wickham, Adv. R)

Specials

"Most functions in R are prefix operators: the name of the function comes before the arguments." (H. Wickham, Adv. R)

infix functions

"function name comes in between its arguments, like + or -" (H. Wickham, Adv. R)

replacement functions

"Replacement functions act like they modify their arguments in place, and have the special name `xxx<-` ... I say they "act" like they modify their arguments in place, because they actually create a modified copy." (H. Wickham, Adv. R)

Infix functions (p. 90)

- Useful to define arithmetic operations
- Example (inbuilt): %*%, %%, %/%
- Example (without %): +, &&, <-, $, @

Infix functions (p. 90)

```
> x<-1
> <-(x,2)
Error:  unexpected assignment in "<-"
> x+1
[1] 2
> +(x,1)
Error:  unexpected ',' in "+(x,"
> `+`(x,1)
[1] 2
> `<-`(x,3)
> x
[3]
```

Infix functions (p. 90)

- User defined infix functions must start and end with %
- Name of function has to be put in **backticks** when defining
- Example :

```
`%+%`<-function(a,b) paste0(a,b)
"new" %+% " string"
> [1] "new string"
```

- `paste0` just concatenates without the separator

roperators package

Assignment operators %+=%

"Modifies the stored value of the left-hand-side object by the right-hand-side object. Equivalent of operators such as += -= *= /= in languages like C++ or Python.
%+=% and %-=% can also work with strings."

For strings: %+%, %-% (string addition and subtraction), %s*%, %s/% (string multiplication and division)

```
> x<-1; x%+=%10 ##11
> "ab"%+%"c" ##abc
> "abc"%-%"b" ##ac
> "ac"%s*%2 ##acac
> "acac"%s/%"c" ##2
```

roperators's manual

Replacement functions (p. 91)

- When defining, replacement function's name has to be put in **backticks** (<- is in name **!**)
- Typically 2 arguments: object to modify, with what to modify
- Additional arguments go in the middle
- Does **not** modify in place, creates copy! **Performance issues**

Copy–paste, `source()` and inside called function
PERFORMANCE!!

`732A94_AdvancedRHT2019_Lecture02_Slide27.R`

Can also behave differently on different `R` build and OS!

Functionals: "apply family of functions"

Higher order functions
Common in mathematics and functional languages

Functionals

Pros

(Often) faster alt. to loops
Easy to parallelize
Encourages you to think about independence (see above point)

Functionals

Cons

Can't handle serially dependent algorithms
Can make code more difficult to read

Common Functionals

`lapply()`
`vapply()`
`sapply()`
`apply()`
`tapply()`
`mapply()`

USE `simplify` argument **!**

Common Functionals: `library(parallel)`

`parLapply()`
`parSapply()`
`parApply()`
`parRapply()`
`parCapply()`
`parLapplyLB()`
`parSapplyLB()`

USE `simplify` argument **!**

Functional programming

"To understand computations in R, two slogans are helpful:

- Everything that exists is an object.
- Everything that happens is a function call.

— John Chambers"

Programming paradigm
Foundation in R
Key abstraction is "the function"
Especially *without side effects!*

R is *not* purely functional, few languages are

Anonymous functions

Functions without names
Often used in functionals

```
sapply(1:n,function(i){i^2},simplify=TRUE)
```

Closures: functions written by functions

"An object is data with functions. A closure is a function with data."
John D. Cook

Closure example 732A94_AdvancedRHT2019_Lecture02_Slide36.R

```
counter_factory <- function(){
  i <- 0
  f <- function(){
    i <<- i + 1
    i
  }
  f ## What is the returned object?
}
## ''function has own parent environment''
first_counter <- counter_factory()
second_counter <- counter_factory()
first_counter()
first_counter()
second_counter()

ls(environment(first_counter))
environment(first_counter)$i
```

R packages

An environment with functions and/or data
The way to share code and data

4 000 developers (date?)
nearly 12500 packages (as of 4 May 2018)

Package basics

Usage

```
library()
::
:::
```

Installation

```
install.packages()
devtools::install_github()
devtools::install_local()
```

Package namespace

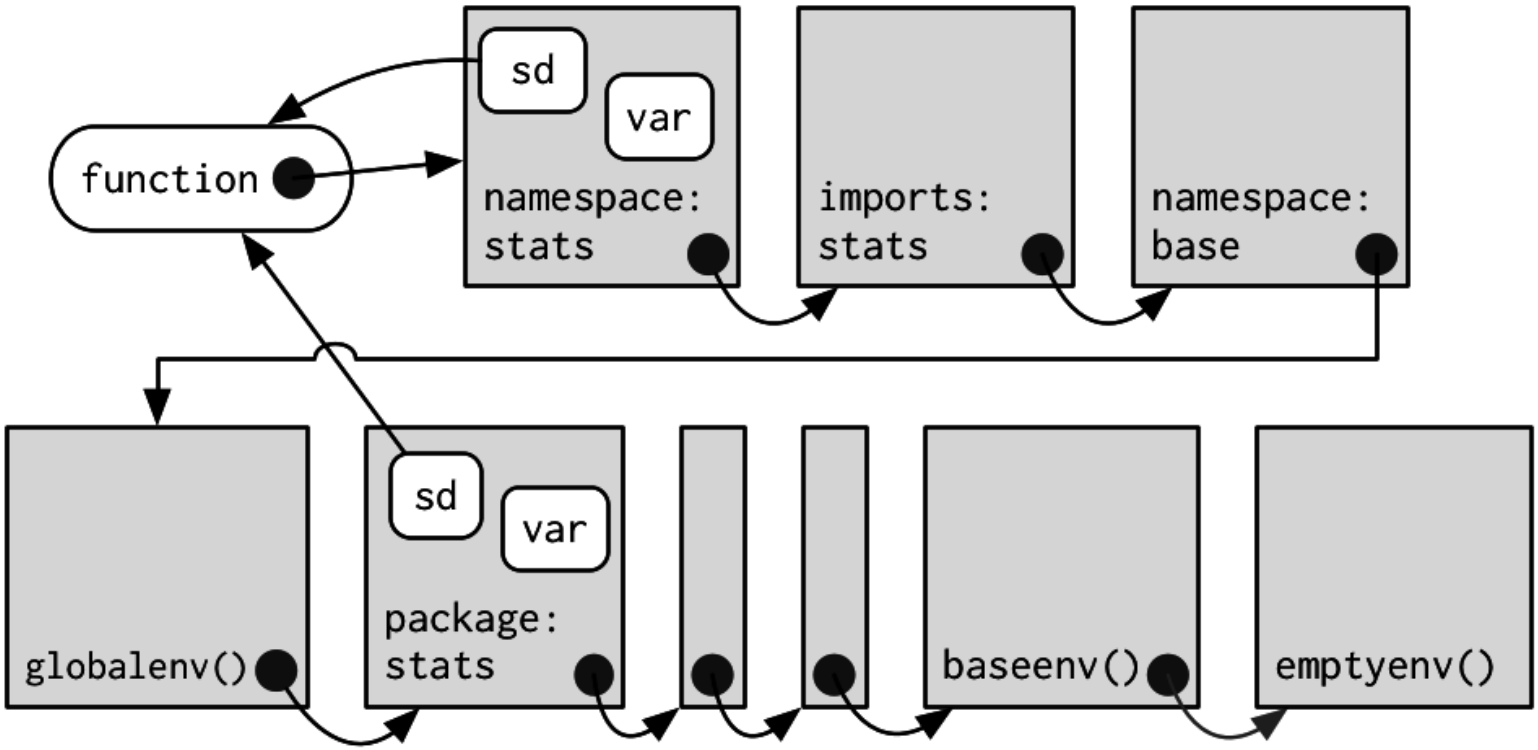

Figure: Package namespace (H. Wickham, Adv. R, p.136)

Which are good packages

Examine the package

1. Who?
2. When updated?
3. In development?

Semantic versioning

"Dependency hell"

`[MAJOR].[MINOR].[PATCH]`

(See reference on course page)

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