

Advanced R Programming - Lecture 2

Leif Jonsson

Linköping University

leif.jonsson@ericsson.com

leif.r.jonsson@liu.se

September 1, 2016

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

Loops

- ▶ for
- ▶ while
- ▶ repeat

See also extra video on program control on course page

For loop

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

While loop

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


Repeat loop

```
repeat {  
# statements  
}
```

Controlling loops

- ▶ break (loop)
- ▶ next (iteration)

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 do R find stuff?)

Current environment ⇒

Parent environment ⇒

...

Global environment ⇒

... along searchpath to...

Empty environment (fail)

Environment search path

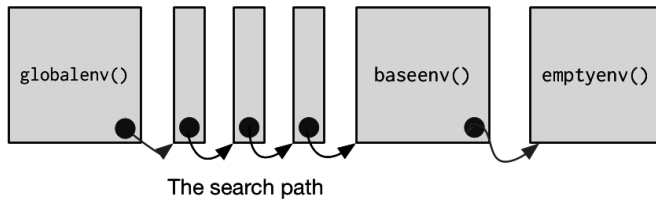


Figure: Environment search-path

Environment basics

"bag of names"

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

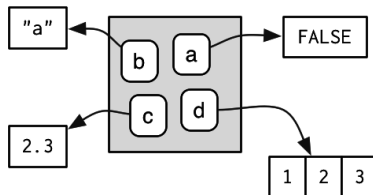


Figure: Environment

Environment relatives

Parents, but no children

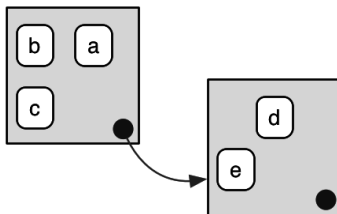


Figure: Env. relations

Working with environments

See environments as lists

```
ls()
```

Assignments

Shallow assignment

`<-`

Deep assignment

`<<-`

Full control assignment

`assign()`

Function arguments

copy-on-modify semantics

specify arguments by...

position

complete name

partial name

Function arguments (cont)

copy-on-modify semantics

```
do.call()
```

```
missing()
```

```
...
```

Default values

Return values

The last expression evaluated in a function

Multiple values using lists

Pure functions

`on.exit()`

`return()`

Specials

infix functions
replacement functions

Functionals

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()
```

Functional programming

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

Closures

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

John D. Cook

Closure example

```
counter_factory <- function(){  
  i <- 0  
  f <- function(){  
    i <- i + 1  
    i  
  }  
  f  
}
```

```
first_counter <- counter_factory()  
second_counter <- counter_factory()
```

```
first_counter()  
first_counter()  
second_counter()
```

```
ls(environment(first_counter))
```

R packages

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

4 000 developers
>7000 package

Package basics

Usage

```
library()
```

```
::
```

```
:::
```

Installation

```
install.packages()
```

```
devtools::install_github()
```

```
devtools::install_local()
```


Package namespace

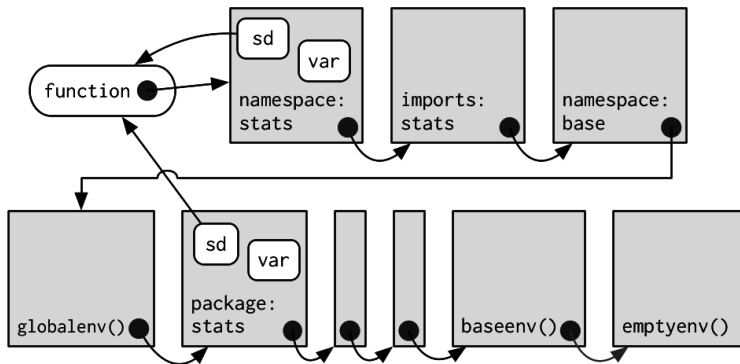


Figure: Package namespace

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!