

roll your own functions

Leo Schalkwyk

June 14, 2012

repetitive operations: scripts

- ▶ you can save a series of commands in a text file
- ▶ run it whenever you like with `source()`
- ▶ practical problem: the environment may change
 - ▶ data available may be different
 - ▶ commands in file may clobber your variables
- ▶ need ways of managing data going in and coming back

Introduction to functions

- ▶ a data type called 'function'
- ▶ has 'arguments' (list of data to go in)
- ▶ has a 'body' (some code)
- ▶ you assign a name just like for any other variable
- ▶ you run it like this : `name()`
- ▶ it has its own environment
- ▶ it returns some results

writing your own functions

- ▶ many of the R functions we've used were written in R
- ▶ you see the source if you forget the ()
 - ▶ try `xor`
- ▶ many functions assign variables
- ▶ why doesn't your environment fill up?
- ▶ why don't your variables get changed?

variable scope and environments

- ▶ there is not just one environment
- ▶ each function has its own environment
- ▶ variables assigned in a function are local
 - ▶ their 'scope' is limited to the function
- ▶ called 'lexical scope' because defined by simple text rule

you can see the enclosing environment (inheritance)

- ▶ almost always better to pass in arguments
- ▶ last thing evaluated in a function is the return value
 - ▶ can be any data type

Arguments: data and options

- ▶ either they are required or they have a default value
- ▶ the first one is usually the data
- ▶ subsequent arguments are often options
- ▶ identified by name or order

A simple function

```
myfirstfunction <-function(x){x-1}
```

- ▶ name of function: myfirstfunction
- ▶ arguments: requires one piece of data
 - ▶ anything evaluating to a number can be used
 - ▶ the name `x` is only used inside the function
- ▶ return value: decremented `x`
- ▶ `x <-100;myfirstfunction(x)`
 - ▶ `x` is still 100
 - ▶ `myfirstfunction` returns 99

the `apply()` function

- ▶ key to processing data by row or column
- ▶ `apply(data,1,mean)` # row means
- ▶ `apply(data,2,mean)` # column means
- ▶ min,max, SD, which.min, summary ...
- ▶ but what if it's a bit more complicated?
 - ▶ often the return value of a function is complicated
 - ▶ the function might not take your data as the first argument
- ▶ these problems are solved using wrapper functions