Functionals in R

Daniel Emaasit

February 28, 2016

Note: This is from *Functional Programming in R* <https://github.com/Emaasit/Functional-Programming-in-R> by Daniel Emaasit

## Functionals

Functions that take a function as input and returns a vector as output. Examples:

* lapply
* sapply + vapply (vector output)
* mapply + Map (multiple inputs)
* apply (matrices & arrays)
* tapply

These functionals are already implemented in base R.

#### Uses of Functionals

* Commonly used as an alternative to **For Loops**.
* For encapsulating common data manipulation tasks like split-apply-combine

#### Advantages using Functionals

* Reduce bugs
* Functionals implemented in base R are efficient & fast

### lapply (for lists)

lapply() takes a function, applies it to each element in a list, & returns a list.

input\_list <- as.list(mtcars)  
output\_list <- lapply(input\_list, length)  
unlist(output\_list)

## mpg cyl disp hp drat wt qsec vs am gear carb   
## 32 32 32 32 32 32 32 32 32 32 32

mtcars[] <- lapply(mtcars, function(x) x / mean(x))

#### Looping Pattern 1: Over elements

Looping over the elements in a list

lapply(xs, function(x) {})

#### Looping Pattern 2: Over numeric indices

Looping over the numeric indices in a list

lapply(seq\_along(xs), function(x) {})

#### Looping Pattern 3: Over the names

Looping over the names in a list

lapply(names(xs), function(nm) {})

### sapply & vapply (vector outputs)

Functionals that take a function, apply it to every element in a list, and return an atomic vector.

* sapply() guesses while vapply() takes an additional argument for the output type
* vapply() is better suited for use inside functions.

df <- data.frame(x = 1:10, y = Sys.time() + 1:10)  
sapply(df, class)

## $x  
## [1] "integer"  
##   
## $y  
## [1] "POSIXct" "POSIXt"

# vapply(df, class, character(2))

### mapply & Map (multiple inputs)

Used when you have two or more lists (or data frames) that you need to process in parallel.

# Generate some sample data  
xs <- replicate(n = 5, expr = runif(10), simplify = FALSE)  
ws <- replicate(n = 5, expr = rpois(10, 5) + 1, simplify = FALSE)  
  
Map(function(xs, ws) {weighted.mean(xs, ws)}, xs, ws)

## [[1]]  
## [1] 0.5896866  
##   
## [[2]]  
## [1] 0.3445823  
##   
## [[3]]  
## [1] 0.5223156  
##   
## [[4]]  
## [1] 0.429464  
##   
## [[5]]  
## [1] 0.5215169

#### Parallelisation (mclapply + mcMap)

Since we can compute each element in any order, it’s easy to dispatch tasks to different cores, and compute them in parallel using the **parallel** package.

library(parallel)  
system.time(mclapply(1:1000, sqrt, mc.cores = 4))

## user system elapsed   
## 0.007 0.013 0.012

system.time(lapply(1:1000, sqrt))

## user system elapsed   
## 0.000 0.000 0.001

### apply (for matrices & arrays)

apply() is a variant of lapply for working hig-order dimensional data objects

m <- matrix(1:100, nrow = 10)  
apply(m, 1, mean) ## MARGINS, 1 = rows & 2 = columns

## [1] 46 47 48 49 50 51 52 53 54 55

apply(m, 2, mean)

## [1] 5.5 15.5 25.5 35.5 45.5 55.5 65.5 75.5 85.5 95.5

### tapply (for ragged arrays)

Ragged arrays are arrays where each row can have a different number of columns. apply() is useful for sumarizing a data set.

## Generate some ragged data  
pulse <- round(rnorm(n = 22, mean = 70, sd = 10 / 3) + rep(c(0, 5), c(10, 12)))  
group <- rep(c("A", "B"), c(10, 12))  
split(pulse, group)

## $A  
## [1] 75 66 67 72 63 66 71 76 68 74  
##   
## $B  
## [1] 75 77 70 81 82 81 75 74 78 76 77 75

tapply(pulse, group, length)

## A B   
## 10 12

tapply(pulse, group , mean)

## A B   
## 69.80 76.75

### the plyr package

### Reduce

Reduce a vector to a single value

Reduce(sum, 1:100)

## [1] 5050

## Find the values that occur in each element in this list  
l <- replicate(n = 5, sample(x = 1:10, size = 15, replace = TRUE), simplify = FALSE)  
Reduce(intersect, l)

## [1] 2 10 3 9

### Predicate Functionals (Filter, Find, Position)

* Predicates are functions that return a single TRUE or FALSE (e.g. is.character)
* Predicate Functionals applies a predicate to each element of a list or data frame.

df <- data.frame(x = 1:3, y = c("a", "b", "c"))  
  
Filter(is.factor, df)

## y  
## 1 a  
## 2 b  
## 3 c

Find(is.factor, df)

## [1] a b c  
## Levels: a b c

Position(is.factor, df)

## [1] 2

### Mathematical Functionals