Functional Programming in R

David Springate

@datajujitsu

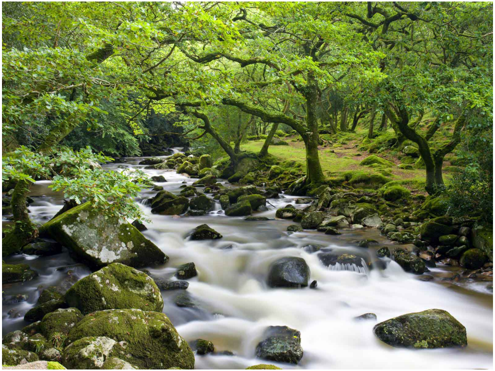
Outline

1. What is functional programming?
2. Elements of functional programming
3. Functional Programming in R
4. A Functional-style generic bootstrap
5. Wrap-up and further reading

What is functional programming?

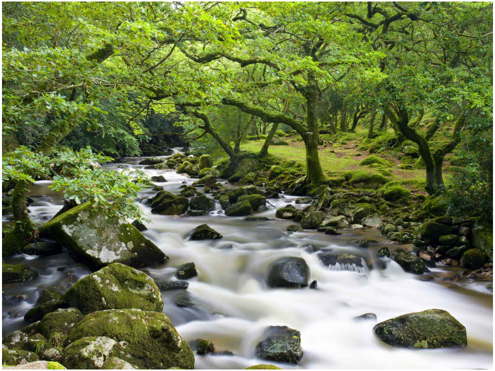
Programming metaphysics

- Programs are representations of reality in a computer
- There are different ways to represent reality. . .



OOP / imperative metaphysics

- C, Python, Java etc.
- Everything is an object with state and behaviour



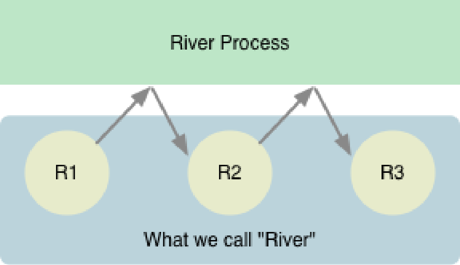
A river is an *object* with various attributes bound to it:

- Flow rate
- Depth
- Pollution levels
- Salinity

Functional Metaphysics

No man ever steps in the same river twice, for it's not the same river and he's not the same man. - Heraclitus

- Lisp, Haskell, F#, Clojure etc.
- *Things* are collections of fixed values which go through processes (functions) over time

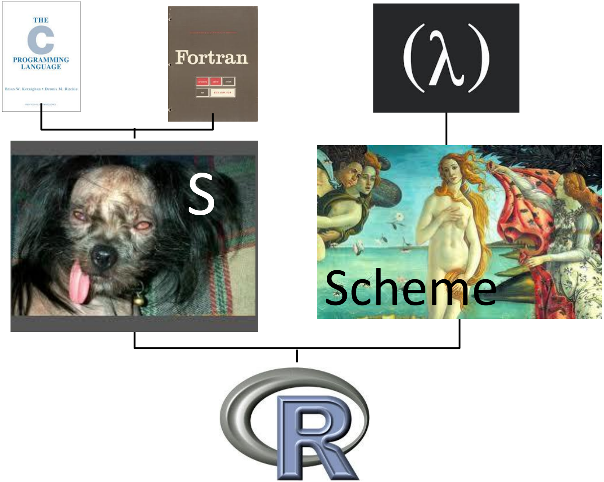


Elements of Functional Programming

1. Functions as first class citizens
2. Vectorised / declarative expressions
3. “Pure” functions - No side effects
4. Anonymous functions
5. Immutability
6. Recursion

Functional programming in R

R Genealogy


THE C PROGRAMMING LANGUAGE
Brian W. Kernighan • Dennis M. Ritchie
Fortran
(λ)
S
Scheme

R is a strongly functional language

. . . **everything** is a function call!

```
> 1 + 2
```

```
## [1] 3
```

. . . is the same as. . .

```
> ‘+‘(1, 2)
```

```
## [1] 3
```

R is a strongly functional language

. . . **everything** is a function call!

```
> 1:10

##  [1]  1  2  3  4  5  6  7  8  9 10
```

. . . is the same as. . .

```
> ‘:‘(1, 10)

##  [1]  1  2  3  4  5  6  7  8  9 10
```

Vectorised functions

Are functions that operate on vectors/matrices/dataframes as well as on single numbers

- Often much faster than looping over a vector
- Higher level - less to debug!
- Very deep in the language

Vectorised functions

Get all even numbers up to 200000

```
> # C style vector allocation:
> x <- c()
> for(i in 1:200000){
+     if(i %% 2 == 0){
+         x <- c(x, i)
+     }
+ }

##    user  system elapsed
##    9.86    0.00    9.88
```

Vectorised functions

Get all even numbers up to 200000

```
> # FP style vectorised operation
> a <- 1:200000
> x <- a[a %% 2 == 0]

##    user  system elapsed
##    0.01    0.00    0.01
```

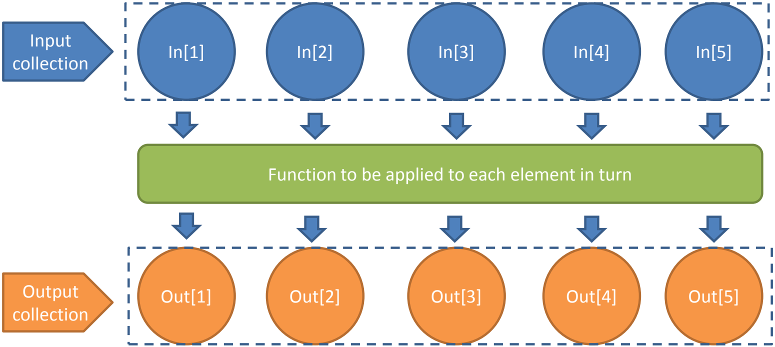
Vectorised functions

Most built-in functions are vectorised:

```
> # e.g.
> paste()
> colMeans()
> rowSums()
> log()
> sqrt()
> x > y
> is.na()
> ifelse()
> rnorm()  # etc. etc.
```

Higher-order functions

. . . Are functions that operate on all elements of a collection (vector/list/vector/matrix/dataframe)



Higher-order functions

You just need to think about what goes in and what you want to come out:

- `lapply` : Any collection -> FUNCTION -> list

Higher-order functions

You just need to think about what goes in and what you want to come out:

- `lapply` : Any collection -> FUNCTION -> list
- `sapply` : Any collection -> FUNCTION -> matrix/vector

Higher-order functions

You just need to think about what goes in and what you want to come out:

- `lapply` : Any collection -> FUNCTION -> list
- `sapply` : Any collection -> FUNCTION -> matrix/vector
- `apply` : Matrix/dataframe + margin -> FUNCTION -> matrix/vector

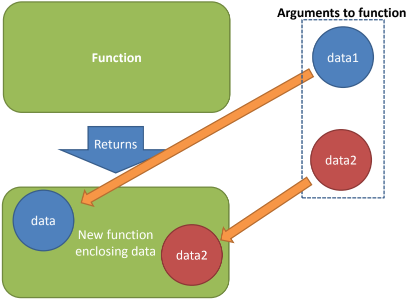
Higher-order functions

You just need to think about what goes in and what you want to come out:

- `lapply` : Any collection -> FUNCTION -> list
- `sapply` : Any collection -> FUNCTION -> matrix/vector
- `apply` : Matrix/dataframe + margin -> FUNCTION -> matrix/vector
- `Reduce` : Any collection -> FUNCTION -> single element

Closures

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



Can build functions that return new functions:

- Useful if some work only needs to be done once, when the function is generated
- Great for optimisation and randomisation problems

An FP-style Bootstrapping function

A Generic function to sample linear models with replacement

Illustrates:

- Functions returning new functions
- Higher-order functions
- Anonymous functions
- Vectorised functions

Will test using the iris data: `data(iris)`
c.f. a non-FP version of the same function

An FP-style Bootstrapping function

```
boot_lm <- function(formula, data, ...){
  function(){
    lm(formula=formula,
       data=data[sample(nrow(data), replace=TRUE),], ...)
  }
}

iris_boot <- boot_lm(Sepal.Length ~ Petal.Length, iris)
bstrap <- sapply(X=1:1000,
                 FUN=function(x) iris_boot()$coef)
```

An FP-style Bootstrapping function

```
apply(bstrap, MARGIN=1, FUN=quantile,
      probs=c(0.025, 0.5, 0.975))
```

```
##       (Intercept) Petal.Length
## 2.5%        4.155       0.3696
## 50%         4.314       0.4072
## 97.5%       4.458       0.4455
```

A Non-FP-style Bootstrapping function

```
boot_lm_nf <- function(d, form, iters, output, ...){
  for(i in 1:iters){
    x <- lm(formula=form,
            data=d[sample(nrow(d),
                   replace = TRUE),], ...)[[output]]
    if(i == 1){
      bootstrap <- matrix(data=NA, nrow=iters,
                    ncol=length(x),
                    dimnames=list(NULL,names(x)))
      bootstrap[i,] <- x
    } else bootstrap[i,] <- x
  }
  bootstrap
}
```

A Non-FP-style Bootstrapping function

```
bstrap2 <- boot_lm_nf(d=iris,
            form=Sepal.Length ~ Petal.Length,
            iters=1000, output="coefficients")
CIs <- c(0.025, 0.5, 0.975)
cbind( "(Intercept)"=quantile(bstrap2[,1],probs = CIs),
      "Petal.Length"=quantile(bstrap2[,2],probs = CIs))
```

```
##       (Intercept) Petal.Length
## 2.5%        4.167       0.3711
## 50%         4.309       0.4093
## 97.5%       4.447       0.4460
```

Wrap-up

Advantages of Functional Programming

Functional programming in R is

- More concise
- Often faster
- Easier to read and debug
- More elegant
- Higher level
- Truer to the language!

. . . than non-fp

Further reading

Functional Programming

mitpress.mit.edu/sicp

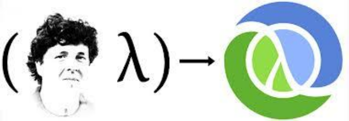
Functional programming in R

github.com/hadley/devtools/wiki

Vectorisation

www.burns-stat.com/pages/Tutor/**R_inferno**.pdf

Programming language metaphysics



http://www.infoq.com/presentations/
Are-We-There-Yet-Rich-Hickey

Thank you

- github.com/DASpringate
- linkedin.com/in/daspringate/
- @datajujitsu
- daspringate@gmail

. . . Any questions?