

Advanced R programming: solutions 2

Dr Colin Gillespie

May 7, 2014

1 Rprofile

1. Create an .Rprofile file. Put in the line

```
cat("Successfully loaded .Rprofile at", date(), "\n")

## Successfully loaded .Rprofile at Wed May 7 21:20:17 2014
```

Restart R. Does the welcome message appear.

2. Try adding my suggestions to your .Rprofile

2 S3 objects

1. Following the cohort example in the notes, suppose we want to create a mean method.

- List all S3 methods associated with the mean function.

```
methods("mean")

## [1] mean.Date      mean.default    mean.difftime
## [4] mean.POSIXct   mean.POSIXlt
```

- Examine the source code of mean.

```
body("mean")
```

- What are the arguments of mean?

```
args("mean")

## function (x, ...)
## NULL
```

- Create a function called mean.cohort that returns a vector containing the mean weight and mean height.¹

```
mean.cohort = function(x, ...) {
  m1 = mean(x$details[, 1], ...)
  m2 = mean(x$details[, 2], ...)
  return(c(m1, m2))
}
```

¹ Ensure that you can pass in the standard mean arguments, i.e. na.rm.

2. Let's now make a similar function for the standard deviation

- Look at the arguments of the sd function.
- Create an function call sd.cohort that returns a vector containing the weight and height standard deviation.²

² Ensure that you can pass in the standard sd arguments, i.e. na.rm.

- Create a default sd function. Look at cor.default in the notes for a hint.

```
sd = function(x, ...) UseMethod("sd")
sd.default = function(x, ...) stats::sd(x, ...)
sd.cohort = function(x, ...) {
  s1 = sd(x$details[, 1], ...)
  s2 = sd(x$details[, 2], ...)
  return(c(s1, s2))
}
```

3. Create a hist method for the cohort class. When the hist function is called on a cohort, it should produce a single plot showing two histograms - one for height and another for weight.

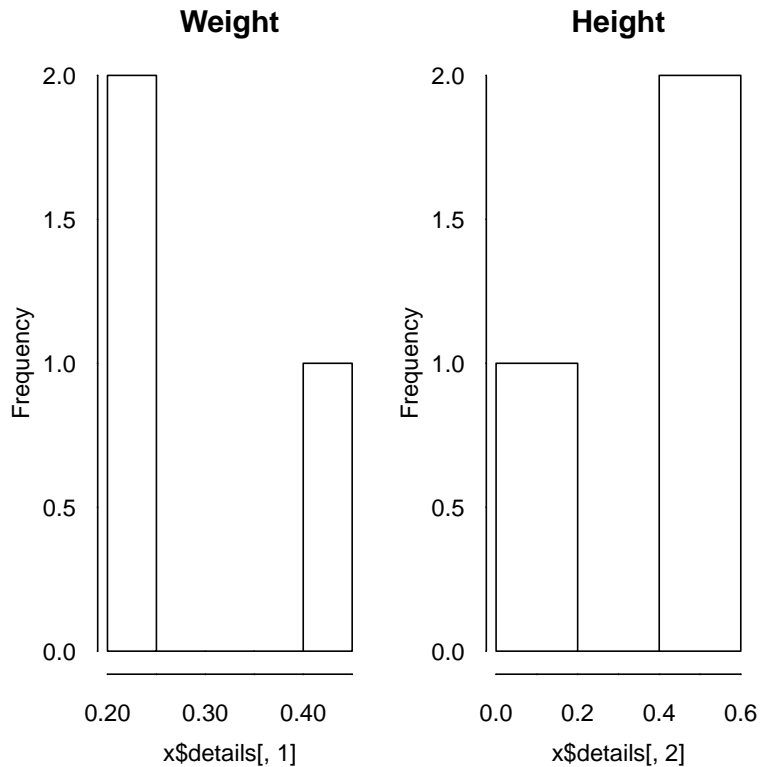
```
## hist is already a generic
body(hist)

## UseMethod("hist")

## Match the args
args(hist)

## function (x, ...)
## NULL

## Function
hist.cohort = function(x, ...) {
  op = par(mfrow = c(1, 2))
  hist(x$details[, 1], main = "Weight")
  hist(x$details[, 2], main = "Height")
  par(op)
}
hist(cc)
```



4. Create a `[` method for the `cohort` class. This method should return a cohort object, but with the relevant rows sub setted. For example, if `c` was a cohort object,

```
cc[1:3, ]
```

would return the first three rows of the data frame.

```
## Lots of methods available.
methods("[")

## [1] [.acf*           [.AsIs
## [3] [.bibentry*      [.data.frame
## [5] [.Date           [.difftime
## [7] [.factor         [.formula*
## [9] [.getAnywhere*   [.hexmode
## [11] [.listof         [.noquote
## [13] [.numeric_version [.octmode
## [15] [.pdf_doc*       [.person*
## [17] [.POSIXct        [.POSIXlt
## [19] [.raster*        [.roman*
## [21] [.simple.list     [.terms*
## [23] [.ts*            [.tskernel*
## [25] [.warnings
##
## Non-visible functions are asterisked

## Examine [.data.frame
```

```
args("[".data.frame")

## function (x, i, j, drop = if (missing(i)) TRUE else length(cols) ==
##      1)
## NULL

"[.cohort" = function(x, ...) {
  x$details = x$details[...]
  x
}
```

5. Create a verb "[" method for the cohort class. This method should allow us to replace values in the details data frame, i.e.

```
cc[1, 1] = 10
```

```
## Lots of methods available.
methods("[<-")

## [1] [<-.data.frame [<-.Date      [<-.factor
## [4] [<-.POSIXct      [<-.POSIXlt   [<-.raster*
## [7] [<-.ts*
##
##      Non-visible functions are asterisked

## Examine [.data.frame
args("[<-.data.frame")

## function (x, i, j, value)
## NULL

"[<-.cohort" = function(x, i, j, value) {
  x$details[i, j] = value
  x
}
cc[1:3, ] = 55
```

3 S4 objects

- Following the Cohort example in the notes, suppose we want to make a generic for the mean function.
 - Using the isGeneric function, determine if the mean function is an S4 generic. If not, use setGeneric to create an S4 generic.

I've intentionally mirrored the functions from section 2 of this practical to highlight the differences.

```
isGeneric("mean")

## [1] FALSE
```

```
setGeneric("mean")

## [1] "mean"
```

- Using `setMethod`, create a mean method for the Cohort class.³ ³ Be careful to match the arguments.

```
setMethod("mean", signature = c("Cohort"), definition = function(x,
  ...) {
  m1 = mean(x@details[, 1], ...)
  m2 = mean(x@details[, 2], ...)
  return(c(m1, m2))
})

## [1] "mean"
```

2. Repeat the above steps for the sd function.

```
isGeneric("sd")

## [1] FALSE

setGeneric("sd")

## [1] "sd"

setMethod("sd", signature = c("Cohort"), definition = function(x,
  na.rm = FALSE) {
  m1 = sd(x@details[, 1], na.rm = na.rm)
  m2 = sd(x@details[, 2], na.rm = na.rm)
  return(c(m1, m2))
})

## [1] "sd"
```

3. Create a hist method for the cohort class. When the hist function is called on a cohort, it should produce a single plot showing two histograms - one for height and another for weight.

```
isGeneric("hist")

## [1] FALSE

setGeneric("hist")

## [1] "hist"

setMethod("hist", signature = c("Cohort"), definition = function(x) {
  op = par(mfrow = c(1, 2))
  hist(x@details[, 1], main = "Weight")
  hist(x@details[, 2], main = "Height")
  par(op)
})
```

```
## [1] "hist"
```

4. Create a `[` method for the cohort class. This method should return a cohort object, but with the relevant rows sub setted. For example, if `c` was a cohort object,

```
cc[1:3, ]
```

would return the first three rows of the data frame.

```
isGeneric("[")

## [1] TRUE

getGeneric("[")

## standardGeneric for "[" defined from package "base"
##
## function (x, i, j, ..., drop = TRUE)
## standardGeneric("[" , .Primitive("["))
## <bytecode: 0x28b1bb8>
## <environment: 0x1918c50>
## Methods may be defined for arguments: x, i, j, drop
## Use showMethods("[") for currently available ones.

## Can you determine what drop does?
setMethod("[", signature = c("Cohort"), definition = function(x,
  i, j, ..., drop = TRUE) {
  x@details = x@details[i, j, ..., drop = drop]
  x
})

## [1] "["
```

5. Create a verb `"[<-"` method for the cohort class. This method should allow us to replace values in the `details` data frame.

```
isGeneric("[<-")

## [1] FALSE

setGeneric("[<-")

## [1] "[<-"

setMethod("[<-", signature = c("Cohort"), definition = function(x,
  i, j, value) {
  x@details[i, j] = value
  x
})
```

```
## [1] "<-"

coh_s4[1, ] = 5
```

4 Reference classes

The example in the notes created a random number generator using a reference class.

- Reproduce the randu generator from the notes and make sure that it works as advertised.⁴
- When we initialise the random number generator, the very first state is called the seed. Store this variable and create a new function called `get_seed` that will return the initial seed, i.e.

```
r = randu(calls = 0, seed = 10, state = 10)
r$r()

## [1] 0.0003052

r$get_state()

## [1] 655390

r$get_seed()

## [1] 10
```

Solutions - see below

- Create a variable that stores the number of times the generator has been called. You should be able to access this variable with the function `get_num_calls`

```
r = randu(calls = 0, seed = 10, state = 10)
r$get_num_calls()

## [1] 0

r$r()

## [1] 0.0003052

r$r()

## [1] 0.001831

r$get_num_calls()

## [1] 2
```

«P2, echo=echo» @

⁴ The reference class version, not the function closure generator.

Reference classes also have an `initialise` method - that way we would only specify the seed and would then initialise the other variables. I'll give you an example in the solutions.

Solutions

Solutions are contained within the course package

```
library("nclRadvanced")  
vignette("solutions2", package = "nclRadvanced")
```