CRAN workshop

Christopher David Desjardins
http://www.github.com/cddesjardins

6 March 2014

Overview

Objects in R (based on Wickham, 2011)

- The R language has three objects systems, S3, S4, and R5.
- S3
  - Not a formal class
  - S3 has been around since R started.
  - S3 is really just a naming convention and a bunch of methods.
  - Very easy to use and most common in R (only OO in base and stats)
- S4
  - Similar to S3 but newer and formal. Rather difficult.
  - `setClass()`, `setGeneric()`, and `setMethod()` define classes, generic functions, and methods. Include slots `@` (for example, see lmerMod)
- R5
  - Suited for simulations that model complex states and GUI. Components that need mutable states
- Can use `pryr::otype(foo)` to determine object type

S3 objects

- Methods for objects of a particular class are called by `method.class()`
- Methods include `summary()`, `anova()`, `print()`, etc.
- Classes include `lm`, `glm`, `factor`, and so on.

```
> library(profileR)
> ls(package:profileR)

 [1] "cp"          "EEGS"        "IPMMc"       "leisure"     "pams"
 [6] "pbg"         "pc"          "pr"          "profileplot" "PS"

> methods(class = "critpat")

[1] anova.critpat*   plot.critpat*    print.critpat*   summary.critpat*

   Non-visible functions are asterisked

> methods(summary)

 [1] summary.aov             summary.aovlist         summary.aspell*
 [4] summary.connection      summary.critpat*        summary.data.frame
 [7] summary.Date            summary.default         summary.ecdf*
[10] summary.factor          summary.funMeans        summary.ggplot*
[13] summary.glm             summary.infl            summary.lm
[16] summary.loess*          summary.loglm*          summary.manova
[19] summary.matrix          summary.mlm             summary.negbin*
[22] summary.nls*            summary.packageStatus*  summary.PDF_Dictionary*
[25] summary.PDF_Stream*     summary.polr*           summary.POSIXct
[28] summary.POSIXlt         summary.ppr*            summary.prcomp*
[31] summary.princomp*       summary.proc_time       summary.profg*
[34] summary.rlm*            summary.srcfile         summary.srcref
[37] summary.stepfun         summary.stl*            summary.table
[40] summary.tukeysmooth*

   Non-visible functions are asterisked
```

More with S3

```
> x <- rbinom(n = 1000, size = 1, prob = .4)
> y <- rnorm(n= 1000,mean=1.4 + .3*x)
> mod1 <- lm(y~x)
> class(mod1)

[1] "lm"

> otype(mod1)

[1] "S3"

> # Can call directly, not advised!
> print.lm(mod1)

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
     1.3428       0.4555

> # Need for Sweave
> cat(try(summary.factor(mod1)))

Error in table(object) : all arguments must have the same length

> stats:::extractAIC.lm(mod1)

[1]  2.00000 41.14482
```

S4 objects

- These are much more complicated
- Best to learn S3 first
- To learn about S4: `http://adv-r.had.co.nz/S4.html`
- A great book for learning about the nuts and bolts of R is: `adv-r.had.co.nz/`
- `lme4` and `Matrix` are S4 packages

Prepping for an R package

- Pull a package off CRAN that you're interested
  - Suggestion: Package by Doug Bates, Hadley Wickham, or John Fox
- Extract the tarball
- Examine the contents
- Use package.skeleton() when you have all the data and R functions you want to include in a package in a clean R environment

What is in an R package?

- DESCRIPTION - description of the package, author(s), and license.
- A `man/` subdirectory of documentation for each R function.
- An `R/` subdirectory of the actual R code.
- A `data/` subdirectory of datasets.
- Maybe a `src/` containing C, C++, or Fortran code.
- `tests/` for validation tests.
- `exec/` for other executables (eg Perl or Java).
- `inst/` for miscellaneous other stuff.
- `configure` script
- CHANGELOG - description of the changes
- NEWS - information about changes in the package
- NAMESPACE - What variables in the package should be exported to make them available to package users, and which variables should be imported from other packages
- A Vignette?

package.skeleton()

```
> trim <- function(x,tr=.1){
+       y=sort(x)
+       n=length(x)
+       qlow=quantile(y,probs=tr,na.rm=T)
+       qhigh=quantile(y,probs=1-tr,na.rm=T)
+       y=subset(y,y > qlow & y < qhigh)
+       trim=mean(y)
+       output <- list(samp.size = n, untrimmed = mean(x),
+                      adj.mean = trim, trim.value = tr)
+       class(output) <- "funMeans"
+       return(output)
+ }
> set.seed(2351234)
> trim.data <- rnorm(n = 10, mean = 5, sd = 25)
> # Create a package called funMeans
> #package.skeleton(name = "funMeans")
```

Read-and-delete-me

```
* Edit the help file skeletons in 'man', possibly
 combining help files for multiple functions.
* Edit the exports in 'NAMESPACE', and add necessary
 imports.
* Put any C/C++/Fortran code in 'src'.
* If you have compiled code, add a useDynLib() directive
 to 'NAMESPACE'.
* Run R CMD build to build the package tarball.
* Run R CMD check to check the package tarball.

Read "Writing R Extensions" for more information.
```

DESCRIPTION

```
Package: funMeans
Type: Package
Title: What the package does (short line)
Version: 1.0
Date: 2014-03-05
Author: Who wrote it
Maintainer: Who to complain to <yourfault@somewhere.net>
Description: More about what it does (maybe more than one
line)
License: What license is it under?
```

DESCRIPTION

```
Package: funMeans
Type: Package
Title: Functions of Means
Version: 0.0.1
Date: 2014-03-05
Author: Christopher David Desjardins <cddesjardins@gmail.com>
Maintainer: Christopher David Desjardins <cddesjardins@gmail.com>
Description: This package will report different means.
License: GPL (>= 2)
```

Licensing your package

- There are several licenses to choose from
  - GNU GPL
    - Free to share, free to modify, free to copy, free to use, free to study
    - Freest license … with a catch!
    - All derivative work must be licensed under the GPL - copyleft
    - You can sell GPL software provided you provide the sources upon request or package them with the binary. Then that person could freely distribute the source code …
  - MIT
    - Permits reuse of your code within proprietary software provided MIT license is included

/R subdirectory

- Can safely delete packageName-internal.R
- trim.R is the script that contains the actual R code
- Let's create a summary method for class `funcMeans`!

summary.funMeans

```
>   summary.funMeans <- function(object, ...){
+       cat("\n Adjusted Mean\n")
+       print(object$adj.mean)
+       cat("\n Trimming Value - ", object$trim.value)
+       cat("\n Untrimmed Mean - ", object$untrimmed)
+       cat("\n Sample size - ", object$samp.size,"\n")
+   }
> funMean1 <- trim(trim.data)
> summary(funMean1)

 Adjusted Mean
[1] 10.86191

 Trimming Value -  0.1
 Untrimmed Mean -  7.757717
 Sample size -  10
```

One more mean function

```
> win<-function(x,tr=.2){
+    y<-sort(x)
+    n<-length(x)
+    ibot<-floor(tr*n)+1
+    itop<-length(x)-ibot+1
+    xbot<-y[ibot]
+    xtop<-y[itop]
+    y<-ifelse(y<=xbot,xbot,y)
+    y<-ifelse(y>=xtop,xtop,y)
+    win<-mean(y)
+    output <- list(samp.size = n, untrimmed = mean(x),
+                         adj.mean = win, trim.value = tr)
+     class(output) <- "funMeans"
+     return(output)
+ }
> funMean2 <- win(trim.data)
> summary(funMean2)
```

```
 Adjusted Mean
[1] 10.53635

 Trimming Value -  0.2
 Untrimmed Mean -  7.757717
 Sample size -  10
```

man/ subdirectory

- This contains all the manpages
- This is time consuming work ...
- Apparently `roxygen2` can help (untested)
- Syntax similar to LaTeX

Vignette

- Why you should create a vignette.
  - It demonstrates how to use your package
  - It allows you to clearly articulate your thinking
  - It can develop into an easy publication for JSS or other similar outlets
- You can include either an .Rnw, .tex, or a .pdf document
- These should be placed in the vignettes/ subdirectory
- Check out `vignette()`

Building the tarball

- There are various ways to do this
  - Rstudio - Use Build & Reload button
  - Terminal - Preferred method (because I use Emacs)

```
chris@debian:~/github/$ R CMD build funMeans/
* checking for file 'funMeans/DESCRIPTION' ... OK
* preparing 'funMeans':
* checking DESCRIPTION meta-information ... OK
* checking for LF line-endings in source and make files
* checking for empty or unneeded directories
* looking to see if a 'data/datalist' file should be added
* building 'funMeans_0.0.1.tar.gz'
```

Checking if it's OK for CRAN

- Again you can do this from Rstudio or a Terminal

```
  chris@debian:~/github/$ R CMD check --as-cran funMeans_0.0.1.tar.gz
* using log directory '/home/chris/github/funMeans.Rcheck'
* using R version 3.0.2 (2013-09-25)
* using platform: x86_64-pc-linux-gnu (64-bit)
 ...
* checking CRAN incoming feasibility ... NOTE
Maintainer: 'Christopher David Desjardins <cddesjardins@gmail.com>'
New submission
...
* checking PDF version of manual ... OK

NOTE: There was 1 note.
```

- This is the only Note you can ignore!
- Any warnings and notes must be dealt with before you submit to CRAN.
- Most messages are insightful!

Submitting to CRAN

- To submit to CRAN follow the instructions here: `http://cran.r-project.org/submit.html`
- Send an email to CRAN@R-project.org from the maintainer address listed in the package using the subject line 'CRAN submission PACKAGE VERSION', where PACKAGE and VERSION are the package name and version, respectively.
- For a new submission, confirm in your email that you have read and agree to CRANs policies.
- Usually in less than 48 hours your package will be on CRAN and binaries will be built for Windows and Macs.

Github

- If you are releasing your software with an open-source license (you are right), then consider developing the package on Github.
- This code is dumped into a funMeans repository
- https://github.com/cddesjardins/funMeans
  - `git clone https://github.com/cddesjardins/funMeans.git`

```
> library(devtools)
> install_github(username="cddesjardins",repo="funMeans")
```

Github

- Great way to share you code
- Git is branching and merging of code is good (supposedly)
- Very quick and easy to get up and running
- Can add other languages too.

How will someone use your code?

- Write and publish your vignette
- Try and get your package included in a CRAN Task View
- Create a website for it (Github can do this)
- Promote it in your signature in your email
- Contribute on Stack Overflow or the R-mailing list

R Users Group

- Interested in more R talk?
- Contact annahelgajons AT gmail DOT com to be added to the mailing list.