Programming in R and Creating Modules

Joey Bernard - ACEnet

What we'll cover

- Programming in R
- Making Modules and Parallel Programming

Programming in R

Programming Basics

All programming languages have some basic similarities

- looping constructs
- conditionals to handle decision making
- the ability to group commands into functions or modules

R has all of these features available

Looping Constructs

There are three types of loops in R

- for loops
- repeat statements
- while loops

Code Blocks

- Before looking at loops, we should look at code blocks
- Normally, R treats each line as a statement and executes it immediately
- Using { and } can wrap multiple lines into a single statement
- R will run a code block at the entry of a new line after the closing }

for loops

For loops have the following form

```
for (name in vector)
  statement1
```

The loop populates ‘name’ with each subsequent value in ‘vector’

On leaving the loop, ‘name’ sticks around with the last value used

repeating

- The simplest form of a loop is the ‘repeat’
- It continues rerunning the statement given until it hits a ‘break’ command
- This is DANGEROUS - easily leads to infinite loops

while loop

- The while loop has the form

```
while (stmt1) stmt2
```

- It loops as long as ‘stmt1’ evaluates to true, as soon as it becomes false the loop ends
- ‘stmt2’ is usually a code block

Controlling Loops

- All common programming languages also have commands to control execution of loops
- next - the ‘next’ command stops the current iteration and begins the next iteration
- break - the ‘break’ command pops execution out of the inner-most loop that we are currently in

Conditionals - if

The if statement has the following form

```
if (stmt1)
  stmt2
else if (stmt3)
  stmt4
else if (stmt5)
  stmt6
else
  stmt7
```

Conditionals - switch

- switch has the form

  ```
  switch (statement, list)
  ```

- If statement is a number, it returns that element from the list
- If statement is a string, it tries to find a match in the list
- The list can be ‘name=value’ sets, where ‘value’ is returned
- If there is no match, NULL is returned

Functions

- All programming languages need some way to reuse code - usually as a function or subroutine
- In R, the syntax for a function is

```
function (arglist) body
```

- To use it, you will need to assign it to a name

```
my_func <- function (....) body
```

Functions - 2

- The arglist is a series of names, which can be used within the function
- Arguments can be given a default value within the definition of the function as a ‘name=value’ pair
-

Modifying the System

- R is very similar to languages like Lisp
- System objects, like calls, expressions and functions can be modified
- This means that almost everything can be altered
- This is very complex behavior, left for the student to research

OS Access

- system - this object runs a command in the host OS, dependent on what the host is
- Sys - this object gets OS information, like the time, environment variables, etc
- file - this object provides file access to copy, move, rename, etc. files

Exceptions - what happened?

- You need to be able to deal with errors
- stop - this command stops all execution and returns to the top-most level and prints out a given message
- warning - this prints out a message, depending on the value of ‘warn’
  - -1 - warnings are ignored
  - 0 - printed at the end of execution (default)
  - 1 - printed right away
  - 2 - turned into errors

Making Modules

Why make modules?

- Both science and open source software grow through sharing
- Once a new process is developed, you can bundle code and documentation together
- With CRAN, you can share your work with people globally

Directory Layout

- An R package consists of a directory containing a file DESCRIPTION and the subdirectories
  - R
  - data
  - demo
  - exec
  - inst
  - man
  - po
  - src
  - tests
- You can also have the file INDEX, NAMESPACE, configure, cleanup, LICENSE, LICENCE, COPYING and NEWS

DESCRIPTION

Package: pkgname

Version: 0.5-1

Date: 2011-01-01

Title: My first package

Author@R: c(person("Joe", "Developer", email = "me@dot.com"),
person("A.", "User", role="ctb", email="you@dot.com"))

Author: Joe Developer <me@dot.com>, with contributions from A.
User <you@dot.com>

Maintainer: Joe Developer <me@dot.com>

Depends: R (>= 1.8.0), nlme

Suggests: MASS

Description: A short (one paragraph) description

License: GPL (>= 2)

URL: http://www.r-project.org, http://www.somesite.com

BugReports: http://bugtracker.com

DESCRIPTION - 2

Mandatory Fields

- Package
- Version
- License
- Description
- Title
- Author
- Maintainer

All other fields are optional

R Subdirectory

- This subdirectory contains only R code files
- Filenames must start with an alphanumeric character, and end with .R, .S, .q, .r or .s
- .R is the most common
- Code should be readable by source(), so any objects should be created by assignments
- Exceptions
  - sysdata.rda - saved image of R objects, lazy-loaded into the same namespace/package environment
  - files ending with ".in" - configure scripts can use these to generate files

R Subdirectory - 2

- You can use R functions to do initialization and clean up
- For packages without a namespace, use .First.lib and .Last.lib
- These are usually put into a file called zzz.R
- .First.lib
  - called with arguments libname and pkgname
  - common use is to call library.dynam to load compiled code
- .Last.lib
  - called with full path to the installed package
  - called just before package is detached
  - uncommon to detach packages, so .Last.lib is rare

man subdirectory

- Should only contain documentation files in R documentation format
- Filenames end in .Rd or .rd
- If you have an empty man subdirectory, it may cause an error during installation
- All user-level objects should have documentation available

- Both man and R subdirectories can contain OS specific files in subdirectories named unix or windows

src subdirectory

- Sources and headers for any compiled code are stored here, along with an optional Makevars or Makefile
- R uses make to compile your code
- Variables and rules for this are determined when R was configured and are stored in R_HOME/etcR_ARCH/Makeconf
- These can be tweaked with macros in Makevars
- If you use a Makefile, be sure you have the targets "all" and "clean"

demo subdirectory

- You can put sample R scripts here, filenames start with a letter and end with .R or .r
- A user can run them by using the command demo()
- These scripts are not run automatically during installation, use tests subdirectory for that
- Should have a file named 00Index, with one line per demo, giving name and short description

inst subdirectory

- Any extra files that you want to have installed as part of the package should be here
- Examples would be extra documentation
- This directory gets installed after the src subdirectory is processed
- This means that the compile step can create files in this subdirectory, which will then get installed
- A common file included here is CITATION, which is used by the citation function

tests subdirectory

- Contains test R scripts that are run at the end of the installation
- The results are written out to .Rout file
- If there is a .Rout.save file, it is compared to the current results and any differences are reported
- Can contain a subdirectory Examples
- If it contains a file named pkgname-Ex.Rout.save, this is compared to results from running the examples

exec subdirectory

- Any additional executable content should be put here
- Typical files are shell scripts, Perl scripts
- Can only contain files, not subdirectories
- These files are automatically set as executable (mode 755) on POSIX systems

data subdirectory

- Contains extra data sets that may be useful to people using your package
- If you have data necessary for your package, it should go into the subdirectory inst/extdata
- Data can be in 3 different formats:
  - plain R code (.R or .r)
  - tables (.tab, .txt or .csv)
  - save() images (.RData or .rda)
- Loaded with the function data()
- Installation can be sped up by including the file datalist
- Tables can be compressed by gzip, bzip2 or xz

Checking Packages

The command check can be used to check your package out

- package is installed, and errors are reported
- file names and directories are checked for validity and permissions
- DESCRIPTION is checked for completeness
- Available index information is checked for completeness
- R files are checked for syntax errors
- Rd files are checked for syntax and metadata
- Any examples are run
- Any tests are run
- A PDF version of documentation is created (to check that Rd files can be converted successfully)

Submitting to CRAN

Before you submit your package to CRAN, you should

- Run "R CMD build" to generate the tarball
- Run "R CMD check" on your tarball
- Check output from examples and tests for errors
- Look for any problems with help file conversions
- Check file and directory sizes, look into compression

When all of your testing is done, upload as anonymous to ftp://CRAN.R-project.org/incoming/ and send an email to CRAN@R-project.org