

# Programming

## 1 Functions

### Session 1 as a script

```
# 1 Import data
mammals <- read.table("c:/projects/day1/mammals.csv", header=T, sep=",", row.names=1)

# 2 Summarize
summary(mammals)
plot(mammals$body, mammals$brain)
plot(log(mammals$body), log(mammals$brain))

# 3 Fit model
mammals.lm <- lm(log(brain)~log(body), data=mammals)
summary(mammals.lm)

# 4 Show fitted line
abline(mammals.lm)
```

### Session 1 as a function

```
session1 <- function(filename.csv)
#####
###
### Function: session1
###
### Purpose: Import data, fit a linear regression model, and plot the results
###
### Args: filename.csv is a comma-separated file with header and 3 cols:
### species,body,brain
### Animal name,1.0,2.0
### ...
###
### Returns: Summary of the regression results (object of class summary.lm)
###
#####
{
  mammals <- read.table(filename.csv, header=TRUE, sep=",", row.names=1)

  mammals.lm <- lm(log(brain)~log(body), data=mammals)
  plot(log(mammals)$body, log(mammals$brain))
  abline(mammals.lm)
  output <- summary(mammals.lm)

  return(output)
}
```

### Use functions, not scripts

Actually, what we did in session 1 is not worthy of a function or a script, in practice we just type `lm()` and `plot()` when we need them. It was a worthwhile session to learn things - every time I learn something new in R, I take notes and store them in a document with other R notes.

Our `cv()` function from session 2 is almost worth keeping, but in practice we just type `sd(x)/mean(x)`. In a typical project, we write functions like `getDistances()`, `plotAreas()`, `tableMonthly()`, `readData()`, and `writeSummary()`. If your function is >40 lines, you may want to split the task into smaller subtasks: `tableMonthly()` calls `getDay()` to process the raw data.

Why functions are better than scripts:

- easy to debug
- easy to change
- more likely to be reused in another project
- focus on each task, often leading to better solutions
- tidy, workspace doesn't fill with temporary objects
- safe, objects are less likely to be accidentally overwritten
- hone your programming skills, for any language

If you have a script, start by converting it to a function() with no args, return()ing some meaningful output at the end, often a list.

## 2 Hints and tips

### Unusual data entries

```

# Identify with:
NA          # is.na(x)
Inf         # is.inf(x)
NULL        # is.null(x)
numeric(0) # length(x)==0
" "         # x==" "
0           # x==0
# See also ?is.nan

```

### Symbols I don't use to create objects

```

# Use instead:
=      # <-
_      # <-
<<-   # assign

```

### Impractical object names

```
T, F # fine in command line, but not in source code (functions or scripts)
```

### Source code format

```

;      # lazy line separator, useful in command line but less useful in source code
{      # braces around clauses (function/for/while/if/else), in separate lines
spaces # spaces help reading, especially when separating top-level args

```

## 3 Import/export data

### Import

Read table in CSV format

```
x <- read.table("c:/temp/mammals.csv", header=T, sep=",")
```

Read data in irregular text format (e.g. 2 numbers in first line, 400 numbers in second line, ...)

```
y <- scan("c:/temp/admb.dat", comment.char="#", quiet=TRUE)
```

Read one line of data

```
y11 <- scan("c:/temp/admb.dat", skip=10, nlines=1, quiet=TRUE)
y26 <- scan("c:/temp/admb.dat", skip=25, nlines=1, quiet=TRUE)
```

### Export

Write table in CSV format

```
write.table(cabbages, "c:/temp/cabbages.csv", quote=F, sep=",", row.names=F)
```

Write vector in one line

```
write(rnorm(10), "c:/temp/admb.dat", ncolumns=10, append=T)
```

## 4 Project management

### What's in a project?

A project in S contains similar things as an elaborate worksheet would in Excel:

- Data                                Vectors and data frames in S
- Results from analysis        Fitted model objects in S
- Plots                                Simple plots like plot(x,y) are easy to recreate, so we don't bother storing those. More complicated plots can be stored as functions without arguments, a.k.a. "scripts", or with default arguments like this:

```
fig1 <- function(x=mammals$body, y=mammals$brain)
{
  plot(log(x,10), log(y,10), xlim=c(-2.5,4.5), ylim=c(-1.5,4.5),
        xlab="Body weight (kg)", ylab="Brain weight (g)", axes=F, pch=16)
  axis(1, at=seq(-2,4,1), labels=10^seq(-2,4,1))
  axis(2, at=seq(-1,4,1), labels=10^seq(-1,4,1))
  box()
}
```

### Option 1 - Get everything out of S

If you don't use S on a regular basis, this can be a reasonable choice

Export: Data as .csv	(or keep them in Excel, Access, ...)
Analysis as .ssc/.r	(showing what was done)
Results as .txt	(or import them into Word, Powerpoint, ...)
Graphs as .eps, .png, .ssc/.r	(or graph the data with some other program)

Then clear the workspace with `rm(list=ls())`

For later use, the source code (.ssc/.r) can be pasted into the command line, or sucked up with the `source()` function.

### Option 2 - GUI management

#### S-Plus

Options - General settings - Startup - Prompt for project folder [v]

Quit S-Plus and start again.

S-Plus now allows the user to select the working directory for that session. It will be saved in the state you leave it in.

To switch between projects, remove garbage objects, quit S-Plus and start again, choosing another working directory.

#### R

When finished working, remove garbage objects and click File - Save workspace

Clear workspace with `rm(list=ls())` # leaves `.First` and `.Last` intact  
 Now quit or load another workspace to switch to another project

It's a good habit to save projects and clear the default workspace regularly (apart from `.First` and `.Last`), to avoid objects with nondescriptive names like `x` and `temp` from accumulating.  
 This way R will start up in <1 sec, ready to start a new project or load an existing one.

### #R: Option 3 - Implement functions

```
.path <- function(project)
#####
###
### Function: .path                                     #
###                                     #
### Purpose: Return full path of project workspace    #
###                                     #
### Args:      project is a string containing project keyword #
###                                     #
### Returns:   String containing full path of project workspace path #
###                                     #
#####
{
  path <- switch(project,
    gmt="c:/programs/gmt/interface/.rdata",
    admb="c:/programs/admb/interface/.rdata",
    sable="c:/projects/sablefish/analysis/.rdata",
    thesis="c:/cwt/thesis/analysis/.rdata")
  return(path)
}

.load <- function(project)
#####
###
### Function: .load                                     #
###                                     #
### Purpose: Load objects from project workspace file into main workspace #
###                                     #
### Args:      project is a project keyword, with or without quotes #
###                                     #
### Notes:     Project keywords are defined in .path and can be updated there #
###                                     #
### Returns:   Invisible vector of object names that were loaded #
###                                     #
#####
{
  load(.path(as.character(substitute(project))), .GlobalEnv)
}

.save <- function(project)
#####
###
### Function: .save                                     #
###                                     #
### Purpose: Save objects in main workspace in project workspace file #
###                                     #
### Args:      project is a project keyword, with or without quotes #
###                                     #
### Notes:     Project keywords are defined in .path and can be updated there #
###                                     #
### Returns:   Null, but workspace file is written #
###                                     #
#####
{
  save(list=ls(1), file=.path(as.character(substitute(project))))
}

```

## 5 Functions categorized by output

In the handouts, I have tried to categorize functions by context. Another perspective is to categorize them by their output:

Those that perform, but return NULL or invisible

`rm`, `plot`, `write.table`

Return an object we'd just like to see

`ls`, `args`, `summary`, `t.test`, `anova`

Return an object we'd like to pass to another function

`log`, `residuals`, `I`

Return an object we might want to store

`c`, `data.frame`, `aov`, `read.table`

## 6 Packages

### Websites

#S: <http://lib.stat.cmu.edu/S/>

#R: <http://cran.us.r-project.org/web/packages/>

### Functions

```
library()
#R: CRAN.packages()
#R: installed.packages()
#R: install.packages("mypackage")
#R: remove.packages("mypackage")
#R: update.packages()
#R: help(package="mypackage")
```

### #R: Recommended R packages

#### Distributed with R

```
lattice    Multivariate plotting
MASS       Datasets and negbin GLM support
mgcv       Simon Wood's implementation of GAM
nlme       Mixed effects models
```

#### Available with `install.packages()`

```
bhat       Nonlinear optimization
car        Applied regression
coda       MCMC diagnostics
Hmisc     Harrell's toolkit
rmeta     Meta analysis
RODBC     ODBC connectivity
spatial    Spatial statistics (kriging and friends)
```

## 7 New functions

---

```
Data manipulation  is.na
                   is.inf
                   is.null
                   is.nan
```

---

	assign
	scan
	write.table
	write
Import/export	scan
	write.table
	write
	#R: load
	#R: save
	#R: CRAN.packages
	#R: installed.packages
	#R: remove.packages
	#R: update.packages
Programming	switch
	substitute
	invisible

---