Automation: Writing Functions

Michael Malick

Updated: 15 May 2013

What is a Function?

A function takes some *inputs* and delivers a specific *output*

- Internal functions (mean, plot, ...)
- Functions in packages (xyplot, melt, ...)
- **User defined functions**

- Functions are R objects with class `function`

Why Write a Function?

- Automate repetitive tasks (reduces errors)
- Modular code that is easier to document
- Extend R's abilities

Function Syntax

```
function(argument1, argument2, ...) {
    step1
       stepA(argument1)
    step2
       stepB(argument2)
    step3 ...
}
```

Example: Hello World

```
hello <- function() {
    print("Hello World!")
}

hello()

hello
```

Example: Square

```
square <- function(x) {
    answer <- x * x
    return(answer)
}
```

```
square(29)
```

- `return()` specifies what the function returns
- If `return()` is not specified the function returns the last line in the function

Example: Clear Console

```
clear <- function(rep = 50) {
    for(i in 1:rep)
       cat("\n")
}
```

```
clear()
```

Example: Standardize

```
stnd.norm <- function(x, na.rm = TRUE) {

    # Standardizes a vector to N(0,1)

    avg <- mean(x, na.rm = na.rm)
    stdev <- sd(x, na.rm = na.rm)
    stnd <- ((x - avg)/stdev)

    stnd
}

x <- 1:10
stnd.norm(x)
```

Example: Linear Model Diagnostics

- script_lm_diag.R

Errors and Warnings

`if` statements are useful for issuing warnings inside functions

```
rep <- 12
if(rep > 10) stop("too many iterations")
```

```
x <- c(10, 9, 8, NA, 7, 6)
if(sum(is.na(x)) > 0)
      warning("NA's were removed before plotting")

x <- x[!is.na(x)]
```

Example: Error Messages

```
boot.median <- function(x, rep = 999) {
     if(rep > 1000) stop("too many iterations")

     res <- rep(NA, rep)

     for(i in 1:rep)
         res[i] <- median(sample(x, replace = T))

     list(boot.mean = mean(res),
         boot.sd = sqrt(var(res)),
         bias = mean(res) - median(x),
         distribution = res)
}

x <- 1:10
boots <- boot.median(x, rep = 1001)

boots <- boot.median(x, rep = 999)
hist(boots$distribution, col = "grey20")
```

Debugging

The `browser()` function is wonderful for debugging

- Interrupts the execution of a function
- Allows for the inspection of the function environment
  - Allows you to call variables defined within the function

Example: `browser()`

```
stnd.norm <- function(x, na.rm = TRUE) {

    # Standardizes a vector to N(0,1)

    avg <- mean(x, na.rm = na.rm)
    stdev <- sd(x, na.rm = na.rm)
    stnd <- ((x - avg)/stdev)

    browser()

    stnd
}
```

- Don't forget to remove the `browser()` call once the function is debugged

Passing Through Arguments

The `...` allows you to 'pass through' arguments to functions called within your function

```
my.plot <- function(x, y) {
    plot(x, y)
}

x <- rnorm(25)
y <- 1:25

my.plot(x, y)
my.plot(x, y, pch = 19) # Fails
```

Example: Pass Through (. . .)

```
my.plot <- function(x, y, ...) {

    plot(x, y, ...)

}
```

```
x <- rnorm(25)
y <- 1:25

my.plot(x, y)
my.plot(x, y, pch = 19, col = 4)
```

You Try...

1. Write a function that computes the mean of a vector
   - Hint: use the `sum()` and `length()` functions
2. Extend the function to plot the input vector
3. Further extend the function to add a horizontal line to the plot that identifies the mean
   - Hint: use the `abline()` function

You Try...

1. Write a function that computes the circumference of a circle (2*pi*r) given the radius as input
   - Hint: use the `pi` function
2. Extend the function to also compute the area of the circle and return both the circumference and area in a list ($pi*r^2$)