Writing Functions Using R

In the following handout words and symbols in **bold** are R functions and words and symbols in *italics* are entries supplied by the user; underlined words and symbols are optional entries (all current as of version R-2.4.1). Sample texts from an R session are highlighted with gray shading.

The Need to Create User-Defined Functions

R, as with any programming language, uses built-in functions to carry out defined tasks. The availability of these built-in functions greatly simplifies calculations. For example, to find the mean for an object requires only the command **mean(***object***)**; thus:

```
> mean(quarters)

[1] 5.583333
```

No programming language, however, includes every possible function that a user might wish to use. When a function does not exist, then the calculation must be carried out using those functions that are available. Suppose that R does not include the function **mean(***object***)**; we can still find the mean by completing the following tasks:

1. determine the number of elements in the object using the function **length(***object***)**
2. add together the elements of the object to get the total sum using the function **sum(***object***)**
3. divide the total sum by the number of elements to get the mean

Here is the code:

```
> n = length(quarters)
> total = sum(quarters)
> Mean = total/n
> Mean

[1] 5.583333
```

Although we can simplify this by combining the first three lines of code into a single line

```
> Mean = sum(quarters)/length(quarters)
> Mean

[1] 5.583333
```

it still provides multiple opportunities for mistyping the code, particularly if we must reenter the code every time we need to find an object's mean. Fortunately, R allows us to define functions of our choosing.

Creating User-Defined Functions in R

Defining a function in R is easy. The general syntax is

name = **function(***argument 1, argument 2…***) {***expression***}**

where *name* is the function's name, *argument* is an object passed to the function (that is, an object on which the function will act) and *expression* is a set of R commands defining the function. For example,

```
> Mean = function(X) {M = sum(X)/length(X); M}
```

defines the function Mean, which takes a single object that is identified as the argument X. The function calculates and prints the object M, which is the mean for the object passed to the function. Now finding the mean is easy.

```
> Mean(quarters)

[1] 5.583333
```

By saving the function as an .RData file, we can use it in a subsequent session after using the command **load(file = "***filename***.RData")**.

Getting More From a User-Defined Function

Our function for calculating and printing an object's mean actually calculates two other properties: the number of elements in the object and the sum of those elements. Having calculated these properties, we might want to know their values. We might, for example, try the following code

```
> Mean = function(X) {      # Because this line ends with an open curly-bracket,…
+ S = sum(X)                # …R recognizes that the code continues on the next line
+ L = length(X)
+ M = S/L
+ S                         # Ask R to print the sum
+ L                         # Ask R to print the length
+ M                         # Ask R to print the mean
+ }                         # The closed curly-bracket means the code is finished
```

Using the function, however, prints the mean only; that is, the function prints only the last requested value. To print all three values we must use the command **print(***object***)** within the function; thus

```
> Mean = function(X) {
+ S = sum(X)
```

```
+ L = length(X)
+ M = S/L
+ print(S)
+ print(L)
+ print(M)
+ }
```

Now our function prints all three values

```
> Mean(quarters)

[1] 67
[1] 12
[1] 5.583333
```

Unless we know the order in which the function prints objects, however, the result is not particularly useful. To improve our function we make the following modifications to our code

```
> Mean = function(X) {
+ S = sum(X)
+ L = length(X)
+ M = S/L
+ s = c("Sum is:", S)          # creates the character string variable 's' in which S is
+ l = c("Length is:", L)       # is converted to a character string; ditto for l and m
+ m = c("Mean is:", M)
+ print(s, quote = FALSE)      # quote = FALSE suppresses the quotes normally placed
+ print(l, quote = FALSE)      # around a character string when printing
+ print(m, quote = FALSE)
+ }

> Mean(quarters)

[1] Sum is: 67
[1] Length is: 12
[1] Mean is: 5.583333
```

There is an important limitation to our user-defined function. Although our function creates the objects sum (S), length (L) and mean (M), and calculates and prints their values, the objects do not exist outside of the function. For example, if we try to use the object S in a subsequent command we get an error message; thus

```
> SS = S*S

Error: Object "S" not found
```

If we wish to use globally an object that we define in a function, then we must assign the function using the symbol <<- in place of the equals sign, allowing us to write a much simpler function; thus

```
> Mean = function(X) {
+ Sum <<- sum(X)
+ Length<<- length(X)
+ MeanValue <<- S/L
+ }

> Mean(quarters)

> MeanValue

[1] 5.58333

> Sum

[1] 67

> Length

[1] 12
```