R and the functional programming model

Before exploring the specifics of R, it would be a good idea to understand the functional model that R is based on.

If you have written programs in C, C++, or Java you used languages based on the procedural model. In the procedural model, a language consists of elements that are combined into procedural instructions that tell the computer, step by step, what to do.

R is based on the functional model. In R the functional model is simple. Everything is either a function or a storage object. A function may have input arguments and it will always return a value. The return value can be displayed in the command console or it can be stored in an object. An objects is either empty or it contains the return value from a function. Interestingly, if you want an object to be empty, you must assign the return value of the NULL function into that object.

The number 4 is a function that has a return value of 4. The equation 2 + 6 is a function that has a return value of 8. The function **mean(x)** in R has an input argument [in this case, the object x is used as the input argument in this example] and it will return the mean of the values contained in x as long as this is a valid operation [R will return an error message if the operation is not valid]. You can also save the return value of a function in an object and use it later. You can do this with the example above using **y = mean(x)**. Now the object **y** will contain the mean of x. If you enter **y**, R will display the mean of x which was stored in y.

R uses two operator symbols to assign a functional return value to an object: = and <-. Using the example above, both **y = mean(x)** and **y <- mean(x)** achieve the same result, the object y will contain the mean of x. You may see either assignment operator in R books or on the internet.

This is a good point in your familiarization to discuss the concept of code style. While the line of code below is acceptable [assuming that the functions and objects are defined],

x = BigFunction(Function1(data1), Function2(data2), Function3(data3))

it is not advisable to use this style. The complexity of the code style, with all of the embedded functions, may make it difficult to understand where you made a mistake if R returns an error message. A more recommended style would separate each function into its own code line, like this:

a = Function1(data1)

b = Function2(data2)

c = Function3(data3)

x = BigFunction(a, b, c)

The return value of each function is assigned to an object and those objects are used as arguments in the last function. This code style will help you understand where the error occurred because R will return an error for the code line that is incorrect. Yes, this simpler code style requires more lines of code, but it is easier to understand what it is doing. Unfortunately, some of the R code examples you may find on the internet will follow the first code style. As you wrestle with them, you may discover why the second code style is preferable.