15.2.1 Fundamentals of R Programming

Now that Jeremy has installed R and RStudio, as well as completed some necessary prerequisites, it's time to start programming. Since Jeremy has learned some programming previously, he decides that the easiest way to learn R is to teach himself how to do the functions he is familiar with. Once he feels comfortable loading in data and manipulating the data within R, he can learn more about the functions in R that will be relevant to his new role.

Now that we understand RStudio's layout and have installed our required libraries and packages, it's time to start programming in R. The two fundamental components to programming in R are creating **data structures** and using **functions.**

When learning a new programming language, it's good to always start with the basic structures and functions you're familiar with, and learn how to implement them using the new language. This way, you can map new concepts and documentation to your previous experience. Once you feel you have mastered the basics of the language, you can always learn the more advanced functionality later.

Just like other object-oriented programming languages, R uses named data structures to store values and properties and uses functions to perform operations. The most straightforward R data structures are named values and vectors.

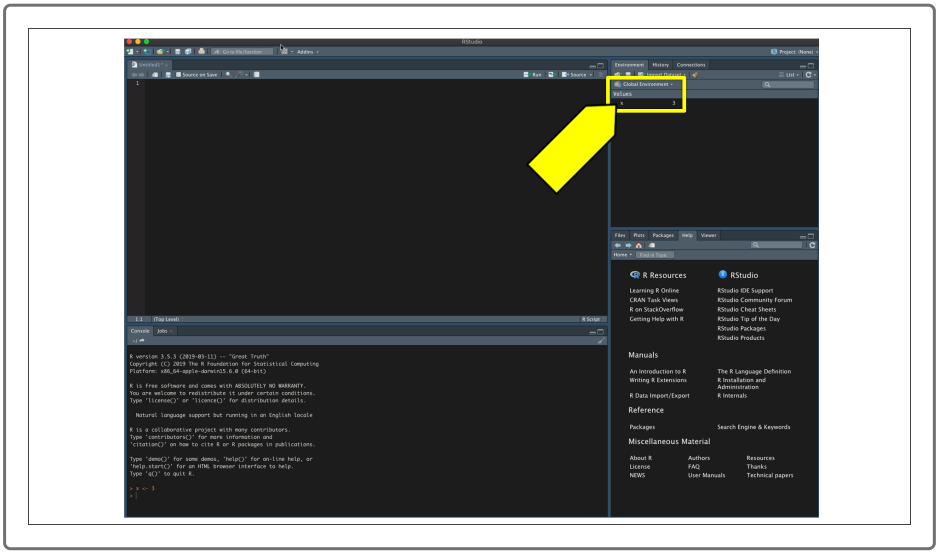
Named values are exactly what they sound like—they are a value that has been given a name. We can think of named values as a variable that has been given a single value. **Vectors** are R's version of arrays, where a list of numbers are assigned a location and stored as a single data structure.

To create our first data structure in R, we use an assignment statement. An assignment statement in R simply tells R the name of the object and assigns a value or data structure to that name.

For example, say we want to create a named value *x* and assign it a value of 3, we would use this R command:

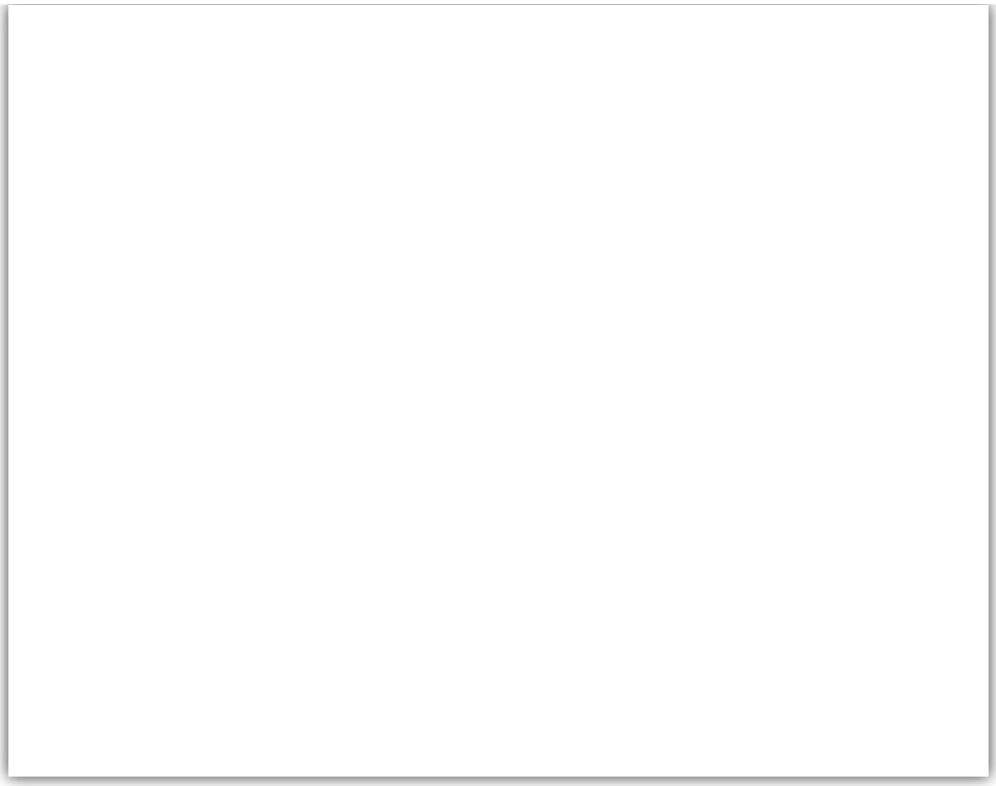
```
> x <- 3
```

The **assignment operator** (`<-`) tells R to assign whatever is right of the arrow to the name that is left of the arrow. In this case, we have given 3 the name of "x." This is similar to assigning a variable in Python, except we use an assignment operator instead of the equals sign (=). Take a moment to run the command in your R console. This next image shows an RStudio session with the newly created "x" value:

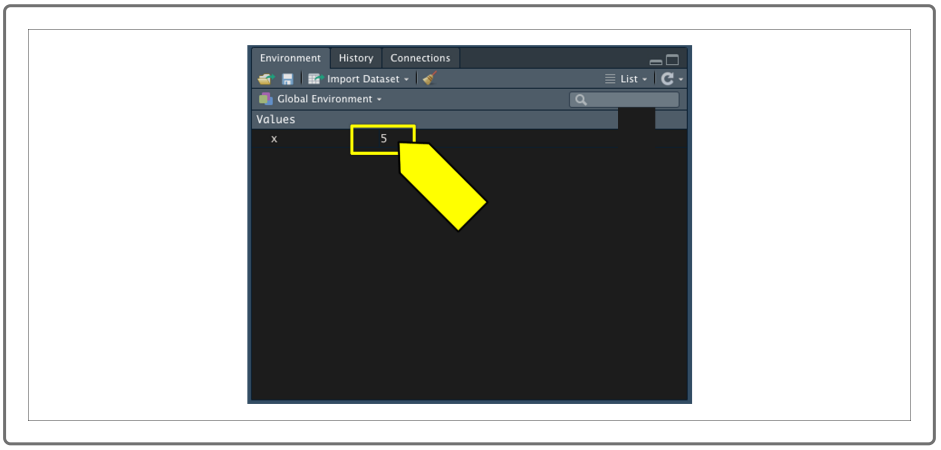


Did you notice that after you created a named value in the console, that named value appeared in your environmental pane? No matter what data structure you use, as you assign objects into your environment, they will appear in this pane. But what happens if you wanted to change that value to something new?

In R, all environment objects are mutable, which means they can be assigned and reassigned multiple times. If we want to assign a new value to *x*, we can do so using an assignment operator to assign a new value.



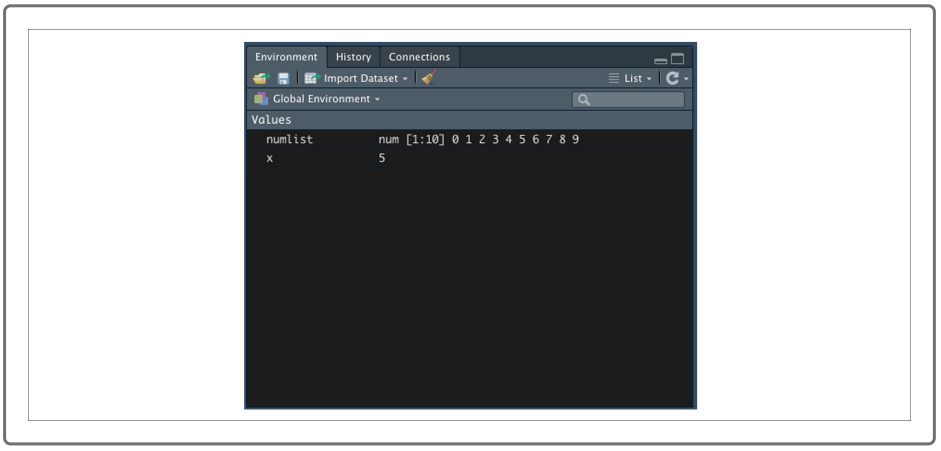
And if we look back to our environment pane, our named value will have been reassigned and the changes reflected in real time:



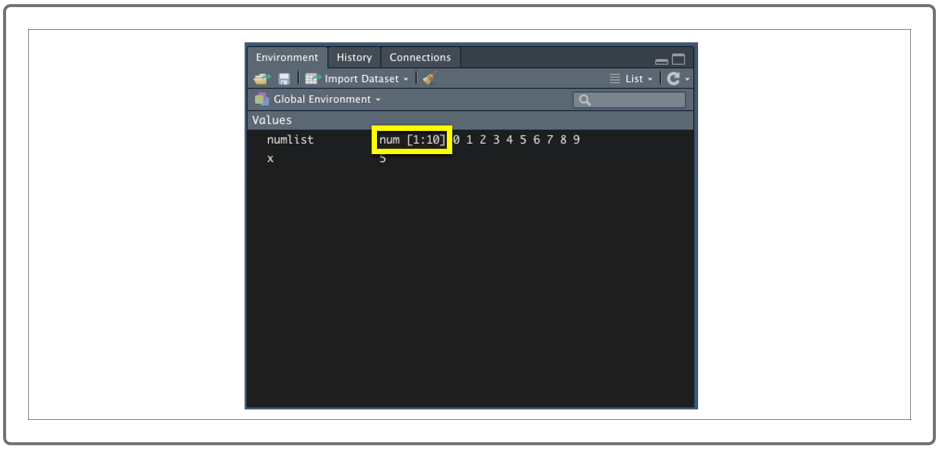
The other simple data structure in R is the numeric vector. A **numeric vector** is an ordered list of numbers, very similar to a **numeric list** in Python. To create a numeric vector, we use the `c()` function. The `c()` function is short for concatenate, which means to link together. In R, we link together a comma-separated list of values into a single numerical vector. For instance, if we want to make a list of numbers from 0 to 9, we can pass the following command into the R console:

```
> numlist <- c(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

The result of this command would assign the vector `numlist` into the environment:



Notice that the environment pane shows the object name. If the object is not a named value, it will provide the data type and dimension. In this case, the `numlist` is a numeric (num) object with one row and 10 columns of values, as shown in the following image:



R also supports a number of more advanced data structures such as **matrices**, **data frames**, and **tibbles**—all of which are variations of the same data frame concept:

- A **matrix** can be thought of as a vector of vectors, where each value in the matrix is the same data type.
- A **data frame** is very similar to a Pandas DataFrame where each column can be a different data type.

- A **tibble** is a recent data object introduced by the tidyverse package in R and is an optimized data frame with extra metadata and features. The most current libraries and packages in R use data frames or tibbles; however, older R packages and analysis scripts will still use matrix objects to perform specific functions or analyses.

Now that we are familiar with assigning data structures and looking at environment objects, it's time to look at the other half of programming in R—using functions.

NOTE

> Those curious about the more advanced R data structures can refer to the **R Introduction documentation (https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf)**.