# Writing Functions in R

**Purpose of writing a function**

There are a few different reasons to write a function. One of the main reasons is that you are writing the same code over and over again. Putting that block of code into a function allows you to make your code neater, easier to read, and faster to edit. Another is the problem of "magic numbers" – values that appear in your code and that have to be manually changed periodically. Writing a function that always produces the correct magic number allows you to run your code without worrying about whether there was a value that needed to be changed.

**How to define a function**

```
Myfunction <- function(parameters){
        Code
        return(value)
}
```

**Parameters and default values**

If your code varies somewhat from one instance to the next, you might have a parameter situation on your hands. Often, I write a very simple function and then discover that there are other use cases that require a few more moving parts. Instead of writing a different function for each situation, I modify the function to allow for the different use cases. If there is a most common use case, set the default value of the parameter so that you don't need to specify it each time.

**View the source code of a function**

To see how a function works, highlight the function and hit F2. This will bring up a view of the source code of the function. If you wrote the function, RStudio should take you directly to the place where the function is defined. If the function came from a package, it will show the function in View Only mode.

**Sourcing a function**

It's a good idea to put your function in a file separate from the rest of your code. To load the function into memory, you can highlight it and run it. However, you can also load the function without ever opening the file in which it is defined, using the source() function. Let's say your function is defined in a file called "myfunction.R", which is located in the current working directory. Simply run source("myfunction.R") and the function will be loaded.

**Using a "functions.R" file in your project**

For some projects, I have a lot of different functions. If they are stored in several different files, it's annoying to run the source() command for each of the files. Instead, you can create a file called "functions.R". Any small functions can be placed in that file. For large functions, put them in their own files, and have source() commands in functions.R to load each of them.

**Printing messages about the status of your function**

It's a good idea to include print() statements in your function to show what the function is doing. That way, if an error occurs, you can see where based on what printed just before the function crashed. However, that can also end up putting way too much stuff in the console, to the point where it is no longer helpful. There are two approaches to this issue: either be frugal in your use of print statements, or include a messageLevel parameter in your function.

Each time you put a print statement in your function, wrap it with an if statement that depends on messageLevel. For messages you almost always want to see, print them only if messageLevel > 0. For messages that you only occasionally need to see, use messageLevel > 1. Have the messageLevel parameter default to 0, and set it in a call to the function if you are doing some debugging.

**Error handling with stop()**

If you want a function to halt at a particular place, use the stop() function. This terminates the function. Every time you run the function and get an error, modify the function to stop before the error and report a useful error message.

**Returning a list of objects**

Sometimes a block of code is used to assign values to more than one object. However, R functions generally assign values to only one object. One way to circumvent this is to wrap up all the things you want to return in a list and then return the list. Then you can use the $ operator to extract the particular object you want to use.