Intro to functions in R

Learning Objectives

After completing this tutorial, you will be able to:

-

What you need

You will need a computer with internet access to complete this lesson and the data that we already downloaded for week 6 of the course.

{% include/data_subsets/course_earth_analytics/_data-week6-7.md %}

> Reproducibilty is actually all about being as lazy as possible! – Hadley Wickham (via Twitter, 2015-05-03)

Efficient coding 101

If you remember from the first week of class, we discussed the concept of reproducibility. A component of reprodcubility had to do with making it easier for your colleagues and your future self to work with your code. This is important, but there is an even more selfish reason to consider creating efficient code.

Efficient coding will make your life easier too.

Don't Repeat Yourself - DRY

DRY (Don't Repeat Yourself) is a principle of software development. The focus of DRY is to avoid repitition of information.

Why?

When you write code that performs the same tasks over and over again, this means, that any edit that you make to even one of those tasks, needs to be made to every single instance of that task! Editing every instance of a task is a lot of work.

`</figcaption>`

Instead, we can create functions that perform those tasks, using sets of arguments

Benefits

Additional benefits:

- Modularity
- Less variables
- Better documentation
- Easier to maintain / edit
- Testing

Modularity

By breaking down your analysis into functions, you end up with blocks of code that can interact and depend on each others in explicit ways.

It allows you to avoid repeating yourself, and you will be able to re-use the functions you create for other projects more easily than if your paper only contains scripts. House of cards vs. house made of lego.

Fewer variables

- Avoid having to track temporary variables: If your manuscript only contains scripts, you are going to accumulate many variables, and you are going to have to worry about avoiding name conflicts among all these temporary variables that store intermediate versions of your datasets but won't need in your analysis. Putting everything into functions will hide these variables from your global environment so that you can focus on the important stuff: the inputs and the outputs of your workflow.
- Keep track of dependencies easily: Functions that produce the variables, results, or figures you need in your manuscript allows you to track how your variables are related, which dataset depend on which one, etc.

Better documentation

Ideally, your code should be written so that it's easy to understand and your intentions are clear. However, what might seem clear to you now might be clear as mud 6 months from now or even 3 weeks from now. Other times, it might not seem very efficient to refactor a piece of code to make it clearer, and you end up with a piece of code that works but is klunky. If you thrive on geekiness and/or nerdiness you might endup over engineering a part of your code and make it more difficult to understand a few weeks later. In all of these situations, and even if you think your code is clear and simple, it's important that you document your code and your functions, for your collaborators, and your future self.

Easier to maintain / edit

If all your analysis is made up of scripts, with pieces that are repeated in multiple parts of your document, things can get out of hand pretty quickly.

Not only it is more difficult to maintain because you will have to find and replace the thing that you need to change in multiple places of your code, but managing documentation is also challenging. Do you also duplicate your comments where you duplicate parts of your scripts? How do you keep the duplicated comments in sync? Re-organizing your scripts into functions (or organizing your analysis in functions from the beginning) will allow you to explicitly document the dataset or the parameters on which your function, and therefore your results, depends on.

Testing

When you start writing a lot of code for your paper, it becomes easier to introduce bugs. For example, if your analysis relies on data that gets updated often, you may want to make sure that all the columns are there, and that they don't include data they should not. If these issues break something in your analysis, you might be able to find it easily, but more often than not, these issues might produce subtle differences in your results that you may not be able to detect.

If all your code is made up of functions, then you can control the input and test for the output. It is something that would be difficult if not impossible to do if all your analysis is in the form of a long script.

Tips for better functions

Easiest way: add comments around your functions to explicitly indicate the purpose of each function, what the arguments are supposed to be (class and format) and the kind of output you will get from it.

Document not only the kind of input your function takes, but also the format and structure of the output.

> input –> function does something –> output

Main benefit: Avoid repetition.

When you write code you should aim for laziness. If you are about to copy a piece of code 5 times and just change a variable in each instance, you are better off converting it into a function. In general, never repeat yourself when you write code, it's called the DRY (Don't Repeat Yourself) principle. Another advantage: Self contained code such that any variable you create inside a function will not be exported into your global environment.