Advanced R Programming

<https://www.coursera.org/learn/advanced-r/home/welcome>

# Week 1 – Functions Overview

Functions are used to **encapsulate** a sequence of expressions that are executed together to achieve a specific goal. A single function typically does "one thing well"---often taking some input and the generating output that can potentially be handed off to another function for further processing. Drawing the lines where functions begin and end is a key skill for writing functions. When writing a function, it's important to ask yourself *what do I want to encapsulate*?

# Week 3

## 3.1 Debugging

This section describes the tools for debugging your software in R. R comes with a set of built-in tools for interactive debugging that can be useful for tracking down the source of problems. These functions are

* browser(): an interactive debugging environment that allows you to step through code one expression at a time
* debug() / debugonce(): a function that initiates the browser within a function
* trace(): this function allows you to temporarily insert pieces of code into other functions to modify their behavior
* recover(): a function for navigating the function call stack after a function has thrown an error
* traceback(): prints out the function call stack after an error occurs; does nothing if there’s no error

1. Running the traceback() function immediately after getting this error:

2. The debug() and debugonce() functions can be called on other functions to turn on the “debugging state” of a function. Callingdebug() on a function makes it such that when that function is called, you immediately enter a browser and can step through the code one expression at a time.

3. The recover() function is very useful if an error is deep inside a nested series of function calls and it is difficult to pinpoint exactly where an error is occurring (so that you might use browser() ortrace()). In such cases, the debug() function is often of little practical use because you may need to step through many many expressions before the error actually occurs. Another scenario is when there is a stochastic element to your code so that errors occur in an unpredictable way. Using recover() will allow you to browse the function environment only when the error eventually does occur.

## 3.2 Profiling

In this section, we will introduce the basics of profiling R code, using functions from two packages, microbenchmark and profvis. Theprofvis package is fairly new and requires recent versions of both R (version 3.0 or higher) and RStudio. If you are having problems running either package, you should try updating both R and RStudio (the Preview version of RStudio, which will provide full functionality forprofvis, is available for download [here](https://www.rstudio.com/products/rstudio/download/preview-release-notes/)).

Once you’ve identified slower code, you’ll likely want to figure out which parts of the code are causing bottlenecks. The profvisfunction from the profvis package is very useful for this type of profiling. This function uses the RProf function from base R to profile code, and then displays it in an interactive visualization in RStudio. This profiling is done by sampling, with the RProf function writing out the call stack every 10 milliseconds while running the code.