Week 2: Introduction to R

Ellen Bledsoe

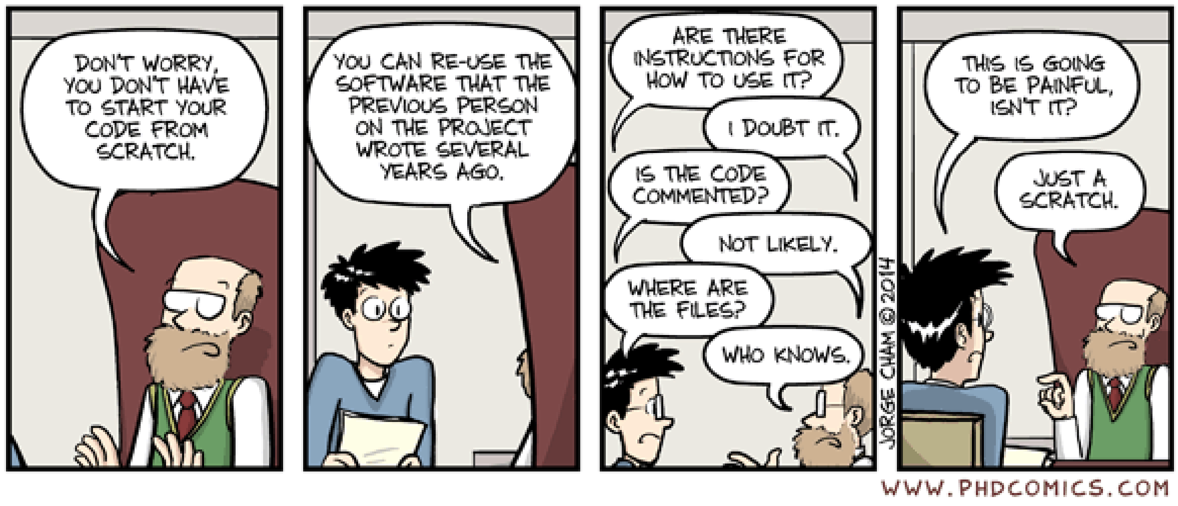
2025-01-28

Introduction to R/RStudio

Why Computer Programming?

As scientists, our goal should not only be to contribute to knowledge but also to share our findings transparently to facilitate thorough peer review and ensure that others can build upon our work.

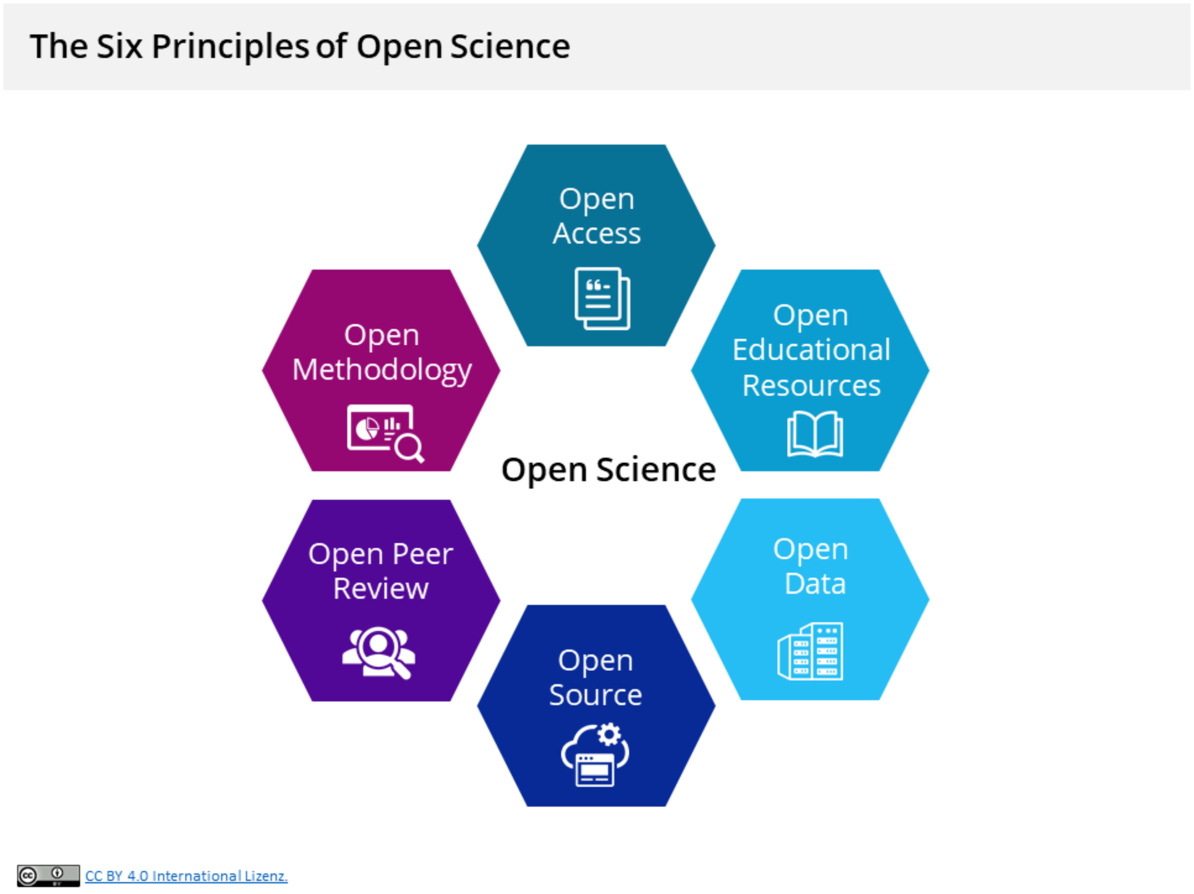
Computer programming can help us achieve these goals effectively by facilitating **computational reproducibility** and contributing to **Open Science**.


DON'T WORRY, YOU DON'T HAVE TO START YOUR CODE FROM SCRATCH.
YOU CAN RE-USE THE SOFTWARE THAT THE PREVIOUS PERSON ON THE PROJECT WROTE SEVERAL YEARS AGO.
ARE THERE INSTRUCTIONS FOR HOW TO USE IT?
I DOUBT IT.
IS THE CODE COMMENTED?
NOT LIKELY.
WHERE ARE THE FILES?
WHO KNOWS.
JORGE CHAM © 2014
THIS IS GOING TO BE PAINFUL, ISN'T IT?
JUST A SCRATCH.
WWW.PHDCOMICS.COM

Computational reproducibility means that all of data processing, analysis, and visualization can be entirely and independently recreated, yielding the exact same outputs. This is particularly important for ensuring the reliability of research, making it easier to verify findings.

Having a clear record of every step taken to process, analyze, and visualize data is necessary for computational reproducibility. Using computer programming instead of "point-and-click" software helps ensure that every step can be recreated.

Computational reproducibility is a large component of **Open Science**, a movement to make research practices, data, methods, and results accessible to everyone. It promotes transparency and collaboration by sharing not just conclusions but the entire process that led to them.


The Six Principles of Open Science
Open Access
Open Educational Resources
Open Data
Open Source
Open Peer Review
Open Methodology
Open Science
CC BY 4.0 International Lizenz.

Open Science has numerous components, as you can see in the figure above. In this class, we are focusing primarily on open data, open methodology, and open source software.

- Using interoperable file types (e.g., CSV files) is an important part of making data openly available.
- Using open-source software for analyses instead of expensive statistical programs (which are often point-and-click) is an important part of making science transparent and accessible.

What's What: R vs. RStudio vs. Posit Cloud

What is R?

- Programming language and software
- Started as a statistics and data analysis environment, but it can also build websites, run simulations, and many other things
- R is what runs all of the code we will write this semester
- R is **open source**, meaning it is free, non-proprietary (not owned by a company), and reproducible. This also means that anyone can contribute to it!
- Separate from RStudio

What is RStudio?

- IDE - Integrated Development Environment
- Makes developing code in R easier by including a number of tools in one place
- Created by a company that is now called Posit

What is Posit Cloud?

- An online version of RStudio that runs in your browser
- We're using it because it:

  1. Avoids installation difficulties
  2. Makes sharing code with instructors for debugging easier
  3. Leaves some of the complexities of working with R until after we've learned the basics

- We will switch to using RStudio on our own computers in the second half of the semester.

Rstudio Mini-Tour

What do all of these panels do?

Let's explore each of the panels in RStudio. They are all useful in their own way!

1. *Source* (upper left): This is where documents which have data or code in them are opened. You can save all the code you type here for future (re)use, which is a big reason coding in R is reproducible.

2. *Console* (bottom left): This is where code from the source is “run” and you see the outputs. You can also execute lines of code which you type into the console, but they will not be saved. You can think of this section as where RStudio really interfaces with R–it is where R actually evaluates code within RStudio.
3. *Environment* (upper right): This panel becomes more helpful as you get familiar with R and RStudio. It keeps track of data objects and other items you have created and gives a bit of information about them.
4. *Files/Help/etc.* (bottom right): This panel is (clearly) very multifaceted. The Files tab lets you see all the files in your current workspace. For us, the Help tab is probably what we will use the most. This is where we can search the R documentation for information about functions we use.

File Types with R Code

Through your experiences using R, you will likely encounter a number of different types of files that contain R code. The two most common are *R scripts* and *Quarto files* (for those familiar with RMarkdown, Quarto is the new and improved RMarkdown).

An **R script** is a plain text file where you write and run R code line by line. R scripts are purely code-based and don’t include formatted documentation for sharing or presenting results. These documents end with `.R`.

In this class, we will be using **Quarto** files (they end with `.qmd`). Quarto files uses something called “RMarkdown” to combine R code with formatted text, allowing you to create dynamic, reproducible documents. This practice is called “literate programming.”

Quarto is a great option for everyday coding but also for creating reports, presentations, or even academic papers and websites. You can integrate explanations, equations, and visualizations alongside your code, and then export the output to formats like HTML, PDF, or Word.

In fact, you will be submitting most of your remaining assignments as PDF files.

Intro to Coding in R

Basic Expressions (R as a Calculator)

Let’s start by typing an expression in the console (below).

Try multiplying 5 and 3 in the Console (*hint*: * means multiply). Hit `Enter` to run that line of code.

Quarto and Code Chunks

As mentioned above, Quarto (`.qmd`) is a file format that lets us incorporate text and code into one document seamlessly. In fact, it is the file format for this document!

- For writing text, you can type as you normally would.
- Code chunks are a bit different:
  - Near the top right of your screen you can toggle between viewing this document as under Source or Visual.
  - If you view this document under `Source` mode, you will see that all code chunks are sandwiched between an opening line of code with 3 backticks and `{r}` and a closing line of code with 3 more backticks.
  - In `Visual` mode, however, you can type R code in the lines under `{r}`.
  - To include text in chunks, you will need to put a `#` in front. R will not read anything in the line after `#` as code.
  - Commenting your code is considered best practice. Not only does it help other people understand what your code is doing, but it will help *you* remember what your code is doing and why!

Code chunks look like this:

```
# This is a code chunk!
```

To run a chunk of code, click the green arrow on the top right corner of the chunk.

You can also run one or a few lines of code at a time by having your cursor on the line or highlighting multiple lines and hitting `Ctrl` + `Enter` (or `Cmd` + `Enter` on a Mac).

A quick shortcut for adding a code chunk is `Ctrl + Alt + i` (`Cmd + Opt + i` on a Mac). Alternatively, you can go to Code > Insert Chunk.

Some Code Chunk Practice

Let's work with an example code chunk.

```
# convert grams to kg
50 / 1000 * 2.2
```

```
[1] 0.11
```

Assigning Objects (Creating Variables)

Assignments are really key to almost everything we do in R. This is how we create permanence in R. Anything can be saved to an object, and we do this with the assignment operator, `<-`.

The short-cut for `<-` is `Alt + -` (or `Option + -` on a Mac)

Note: You can technically use either `<-` or `=`. I recommend getting used to `<-`, as this is standard practice (now) in R and recommended in style guides. The reasons why are complicated and don't often arise in most situations, but it is still good practice.

```r
weight_g <- 50
```

We can see that this object has been created by looking in the Environment tab.

Objects work just like the value itself. For example, we can use them in mathematical expressions.

```r
weight_g / 1000
```

```
[1] 0.05
```

```r
weight_g / 1000 * 2.2
```

```
[1] 0.11
```

```r
weight_lb <- weight_g / 1000 * 2.2
```

The object won't change unless you assign a new value to it directly using the assignment operator (`<-`).

```r
weight_g
```

```
[1] 50
```

```r
weight_g * 2
```

```
[1] 100
```

```r
weight_g
```

```
[1] 50
```

```r
weight_g <- 26
weight_g
```

```
[1] 26
```

Some RStudio Tips

Use the `tab` key to auto-complete objects (and other things)

- Let the computer do repetitious work.
- It's easier and causes fewer mistakes!

Also, you can see the commands you've run under `History` in case you forgot to write something down in your Source code.

Let's Practice

Head over to this week's assignment. It is already in this Posit Cloud project, so you can find it in the Files tab. I recommend clicking over into Visual mode if it isn't already there.

Start working on Question 1 and 2 in the Assignment. Remember to comment your code as you go!

Functions

Functions are pre-written bits of codes that perform specific tasks for us. Effectively, they are complex expressions rather than "basic" ones.

A function call consists of two main parts.

1. The name of the function, followed by parentheses.
2. Arguments, which are pieces of information that the function requires to perform its task. Anything you type into the parentheses are arguments.

For example:

```r
sqrt(49)
```

```
[1] 7
```

Here, `sqrt()` is the name of the function, and `49` is the argument.

We can also pass objects as the argument in functions.

```r
weight_lb <- 0.11
sqrt(weight_lb)
```

```
[1] 0.3316625
```

Data Types

Another useful function is `str()`, which is short for "structure."

It will tell us information about the dimensions and data type of an object.

```r
# numeric
str(weight_lb)
```

```
 num 0.11
```

Another data type is for text data. We write text inside of quotation marks.

```r
"hello"
```

```
[1] "hello"
```

If we look at the structure of some text we see that it is type "character."

```r
str("hello world")
```

```
 chr "hello world"
```

Many functions can take multiple arguments. For example, let's use a function to round `weight_lb` to one decimal place.

```r
round(weight_lb, 1)
```

```
[1] 0.1
```

Functions return values, so as with other values and expressions, if we don't save the output of a function then there is no way to access it later.

It is common to forget this when dealing with functions and expect the function to have changed the value of the variable. However, if we look at `weight_lb`, we see that it hasn't been rounded.

```r
weight_lb
```

```
[1] 0.11
```

To save the output of a function we assign it to a variable.

```r
weight_rounded <- round(weight_lb, 1)
weight_rounded
```

```
[1] 0.1
```

Let's Practice

Start working on Exercises 4 and 5.