**Caffeine case study**

An introduction to R and professional programming practices using a case study on caffeine advertising. Adopted from Carl Bergstrom and Jervin West [B&W] by Dr. Diaz Eaton, Bates College, for use in DCS 105. <https://callingbullshit.org/case_studies/case_study_caffeine_free.html>

Learning Objectives:

* Terminology related to programming
* Professional programming practices
* Practice R syntax and RStudio environment

**Programming** is a broad term that encompasses a process including elements such as planning, implementation (**coding**), testing, revision, feedback, etc. **Design & Modeling** also encompass these elements but are terms that originate from different subject domains.

We can follow B&W’s work with pen and paper to solve. We can also “code this up.” That way if we had different estimates for any parameters (such the caffeine content in another brand of coffee), we wouldn’t have to completely recompute!

**PLAN**

**\* Coding professional practice - Intentionally plan your code before you start typing.**

From our case-study: **“**To figure out how much caffeine is in an ounce of coffee:

Center for Science in the Public Interest, there are [415 milligrams](https://cspinet.org/eating-healthy/ingredients-of-concern/caffeine-chart) of caffeine in a 20oz Starbucks coffee. (This turns out to be on the strong side relative to other drip coffees or milk-based espresso drinks.) That corresponds to about 21 mg of caffeine per ounce. An ounce of water weighs about 28 grams[\*](https://callingbullshit.org/case_studies/case_study_caffeine_free.html#footnote). Thus a Starbucks drip coffee is about 0.075% caffeine by weight. **In other words, strong coffee is also 99.9% caffeine free!”**

In order to figure out how caffeine-free coffee is, we need to/B&W did:

1. Calculate much caffeine (mg) is in a ounce of coffee.
2. Calculate the mass (in mg) of an ounce of coffee.
3. Divide mass of caffeine/total mass to get the proportion of caffeine prepared coffee & multiply by 100 to get the percentage caffeine.
4. Subtract from 100% to figure out how “caffeine-free” prepared regular [Starbucks] coffee is.

Above, I am following the process that B&W did. There is more than one way to get to the answer without doing these particular steps, and that is okay. **Programming is often a creative process.** When there are multiple ways to approach a programming challenge, you might write out a few plans, and choose the one that either has the least amount of steps, the one that is easier to read/follow by another user, or one that is easy to expand upon later. It often will depend on your overall goals.

In the absence of a guide like B&W’s solution above, it is common to start at the end, figure out what pieces of information you need, and work backwards. Ex/How to I get a %? Well, I’ll need to know the caffeine mass and divide it by the total… etc. The technical term for this is “**Backward Design.**”

**IMPLEMENT**

**\*Coding professional practice - name the parameters close to what they mean. Use # (aka “comment out”) before descriptors that explain what you are doing and/or the units, or where the figure comes from.**

Note that the same rule from above applies - the coding process is also a creative process with multiple ways of getting the same output, so consider your goals when choosing your approach. When you implement code, you want this reasoning to be understood by others, so you have “comment” to explain. Why bother?

Extrinsic motivation: You need me to follow your code to grade it.

Altruistic motivation: Often, code is used by other people and is built on later. They cannot build on what they do not understand. While these exercises are smaller, the intention is for you to build good practice now that respects your future colleagues.

Selfish motivation: Most of the time, the person who comes back and builds on prior code is YOU. And if it has been a few weeks or a few months since you last used your code, you will probably not even remember what you did, where you got that parameter, why you chose to implement in that way, etc. So think about being respectful to your future self colleague.

Go to **File > New File > R-script** to get a workspace on RStudio that is similar to the DataCamp workspace. We are going to be working in the script, which will allow you to save a set of commands as one program that can be run at a later time.

**\*Professional programming practice - in the first line of the script add “#” with file info (name, author, what this is for)”**

#Diaz Eaton and Hanson Shrout, DCS 105 A Calling Bull

#Caffeine Case Study, Jan 23, 2019

**Step 1:**

#Calculate how much caffeine (mg) in an ounce of coffee

CaffinCup <- 415 #milligrams (mg)

OzinCup <-20 #oz

CaffinOz <-CaffinCup/OzinCup #mg/oz

#Should get 21 #mg/oz

**\*Coding professional practice - build in ways to check your code to make sure it is functioning as intended**

**Step 2:**

**#**Calculate the mass (in mg) of an ounce of coffee.

ginOz<-28 #g/oz

mginOz <- 1000\*ginOz #mg/g \* g/oz = mg/oz

**Step 3:**

**#**Divide mass of caffeine in 1 oz/total mass in 1 oz to get the proportion of caffeine prepared coffee & multiply by 100 to get the percentage caffeine.

PerCaff <- 100\*CaffinOz/mginOz #100%\*mass of caffeine/total mass

**Step 4:**

#Subtract from 100% to figure out how “caffeine-free” prepared regular [Starbucks] coffee is.

CaffFree <- 100 - PerCaff #Gives % caffeine-free

**ANALYZE**

What result did you get? Same as J&W? Different? Is the code working properly?

**MODIFY**

1. **Discuss & Plan:** How would you modify the code you wrote in the script to check the caffeine-free % claim of the Nestle cocoa?

“most cocoas have about 20mg of caffeine in a 8 ounce cup,”

1. **Implement**
2. **Analyze output:**
	1. What answer did you get?
	2. Can you verify Nestle’s claim that it is 99.9% caffeine-free?
	3. Can you verify J&W claim “...it is 0.009% caffeine by weight.”?

**CHALLENGE**

How else might you code this problem that would be more efficient?