1 Some Principles of Writing Code

Why do we give you such a specific design recipe to follow? The answer to that question lies with two facts about programs and programming:

1. You’ll spend more time debugging than writing most of your programs.
2. Programs are not only read by computers: They are also read by humans!

Following the design recipe will help you to be clear—in your own mind and in your comments—about exactly what each procedure you write accomplishes. The clearer your thoughts and comments, the clearer your code; the clearer your code, the less buggy it will be, the less time you will spend debugging (i.e., banging your head against the wall).

Code, by the way, is just what programmers call the programs they write. Bugs are mistakes that keep code from doing what it’s intended to do. Debugging is the process of trying to first locate, and then “squish” bugs in order to get your code working as intended.

1. Write beautiful code. You should aim for beautiful code. This means, among other things, using idioms and following formatting conventions. If you do, your TAs will understand your code, and award you lots of points, and you will understand your code when your future self revisits it weeks, months, or years later.

2. Don’t be too clever. Debugging a program is much harder than writing a program. You should write simple programs, so that the debugging process goes as smoothly as possible. And you should make your code easy to understand. A general rule of thumb is that debugging \( n \) lines of code takes \( n^2 \) work.

3. Get the design right, then start coding. Don’t write a program that gets at the general idea but leaves out pieces and then mess around with it later until it does what it’s supposed to. Instead, think about your entire program before you jump in and start coding. This will help you avoid writing ugly code.

4. Don’t be afraid to throw code away and start over. You may realize part way through a program that there is a better way to do what you are trying to do. If this is the case, start over! Trying to adapt to a new idea in the midst of programming or stubbornly
plugging along with an old way when you know a better one is a recipe for disaster. Remember: You’re better off writing a piece of code ten times than debugging it once.

Caveat: Instead of just deleting code you are “throwing away”, comment out that section of your program so that you can come back to it if you eventually decide that it might be useful.

5. **Simplicity counts.** Nick Trefethen has written a beautiful book called *Spectral Methods in MATLAB*. In the introduction, he says,

The...programs in this book are terse. I have tried to make each one...fit on a single page, and most often on half a page. Of course, there is a message in this style...: you can do an astonishing amount of serious computing in a few inches of computer code. And there is another message, too. The best discipline for making sure you understand something is to simplify it, relentlessly.

Follow Nick’s advice and you won’t go wrong. You may even hear Spike use his name as a verb: “Let’s Trefethen this code!”

Sometimes, it will seem like you have to choose between two desiderata: simplicity and efficiency. Efficiency is very important, but as a rule of thumb, don’t sweat the small stuff. Saving a few operations here and there won’t make much of a difference. On the other hand, writing code that you can understand later will save you a lot of time in the long run.

6. **Elegance pays.** Elegant code will make your life wonderful. It is easier to catch bugs, you can extend it to do more if necessary/desired, and the TAs will love you for it!

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS17 document by filling out the anonymous feedback form:

[http://cs.brown.edu/courses/cs017/feedback](http://cs.brown.edu/courses/cs017/feedback)

---

*In CS 17, we’ll discuss efficiency (*ad nauseam*), so soon you’ll have at your disposal the tools to measure how much more efficient one procedure is than another.*