**Coding is Thinking**

Most of the courses I teach in our major make use of computation, but they are at different levels, so there are different, but related, challenges to each.

In the sophomore-level course I teach on computer modeling of physical systems, the students are having their first experience away from introductory-level problem sets. They are getting into open-ended problems for the first time, as well as seeing that there can be a number of ways to investigate, evaluate, and present potential solutions to problems. They are also seeing for the first time what it means to think about models as having *potential* to solve a problem and having to make a decision or recommendation about what is appropriate and why. Finally, they are also seeing that it is useful to see a range of predicted values for a model, and how scientists go about using those ranges to think about problem solving or where models might break down. In the second sophomore-level course I teach, on data analysis and presentation, the students are getting experience in data collection and management as well as making choices in data presentation. They are often shocked to see how many ways you can present findings from the same dataset, or how subtle choices can help tell a data story or improperly hide findings. Finally, in my upper-level analytical mechanics course, they are proposing models to explain physical actions described only in text and developing ways to assess the model validity.

In my experience, I find that students are reluctant to commit to code – to actually type things out – in a brainstorming sort of way. Once they’ve done it, they then see themselves as stuck with what they have done and are reluctant to change it or seek help. They think that they must write the perfect code the first time. In some ways, I am still personally mystified by this. But I try to convince them that just as writer David McCullough has said, “writing is thinking. To write well is to think clearly. That’s why it is so hard,” coding is thinking. And thinking involves brainstorming, trying things out, erasing or commenting code, talking with others, sketching things out on paper, etc. A final product of code should represent lots and lots of thinking and probably will be the result of outlining, reading literature, talking with others, and just plain thinking, not to mention sorting out and responding to error messages. I want them to see coding as a fluid process of continual development and improvement, a wonderful slate where ideas can be worked out and aha moments come through.

To get students to this point, we do a lot of short-time-limit challenges in class: “see what you can code, in ten minutes, for answering this question.” Then we share a lot. And we share a lot of code-in-progress, not just final versions. In some ways, probably actually many, I model my class time and projects after some of the writing across the curriculum techniques I have learned from my friends in English: write (code) a lot and check in with others frequently.