Some questions have popped up recently that I thought would be worth answering broadly. The following represents my **personal opinions** from 30 years of teaching CS so take it for what it’s worth. It is offered as a kind of FAQ. As background, CSE 231 has around 700 students a semester, CSE 232 about 400/semester. If you have an opinion, offer it up, we’d love to hear from you.

**Lectures**

**Lectures are important.** Less so than you think. My typical experience with a 232 class and 200 students (lecture twice as week) was: **at best** 50% attendance to the lecture. In the lecture **at best** 50% actually paying attention (asking questions, trying stuff in class). So maybe I was talking to 25% of the class on a good day. Why so low? There are plenty of studies which show an 80 minute lecture is boring and students lose focus (80 minutes of talking about code can be deadly boring). When we provided videos and lecture notes, many felt they didn’t need a lecture. Short videos on the order of 10 minutes are easier to digest and folks could do the work when they had time. The more we did that kind of stuff, the fewer the number of students that showed up for lecture.

**But lectures allow us to ask questions.** True, and I really enjoy the questions and talking about topics. Lecturing can be as boring as listening if no one is engaged. But, for the most part, students in a big lecture don’t ask questions, no matter what I do to encourage it. Most students come to the lecture unprepared (haven’t looked at the material) and so are just hoping to soak something up.

**The material isn’t being as well covered without lectures.** Not really. Even though the class has grown to 400/semester, the grade averages and distribution have remained at least the same, if not slightly better, using the online approach. The class covers the same material, has similar exams (same material), similar projects (many recycled from earlier) as compared to when lectures were offered.

**I want a lecture anyway!** OK, but given the above the administration may not feel it is a good use of resources to provide multiple lecture sections (at least 2 for 232, at least 3 for 231). By the way, these decisions were not made by the lecturers but by the administration for the reasons listed above. I like teaching, I enjoy talking to students. I’m sorry I don’t get to know students well anymore. But there are a lot of you now as compared to even 4 years ago. We let everyone in who wants to give it a try, but there are compromises that have to be made at these class sizes.

**Projects**

**Why a project every week?** Two things. First, the only real way to learn how to program is to do it. Talking about programming is helpful, but actually programming is how you really learn. Second, and this is not a criticism but a fact, many students (half??) wait until the last minute to even start the work. Waiting two weeks to try out new programming concepts likely means you aren’t getting that practice.

**Project specs are unclear.** I struggle with this. We really do try to write out in detail what we want for the individual pieces of a project. Specifications do not tell you how to do something, only what is required for the program to do. The challenge, the hard part, is solving the problem, i.e. the “how.” If we told you how to do each and every step, it would be trivial to write the program. That’s on purpose, we want you to work this stuff out on your own.

**I worked XX hours and I didn’t finish.** I sympathize with this, but what is the solution? Learning how to spend your time effectively is a skill you learn in college. We try to break things down, give you a “ladder to climb” in the sense of program pieces to address, and then let you climb the ladder (solve more pieces) as you have time. In this way we hope to help you be more effective.
Projects take too long, are too hard. Maybe? We know that we have run these projects, in different forms (so they cannot be copied), multiple times with similar results. We also know that, at least by submission on Mimir, about ½ the class has not yet started by Sunday, and about ⅓ not started by Monday. We hope folks are developing locally but that is usually not the case. It is hard to finish in a marathon session, easier when you give yourself time.

Help

Piazza sucks! It is not a substitute for a person but it is what we have for 24/7 access. It also allows other students to help. It is often the case that the same question gets asked over and over again and the answers will tail off for the same question. It is also the case that if an answer has been offered, then no one else will look to provide a different answer, or at least not often. Given the volume it is a decent, though not perfect, solution.

Helproom sucks! Well, I can say that Monday night helproom is a zoo and the TAs, by my direction, are told to get to as many students as possible. That means less help per student, but more students helped. Monday night is for clean-up, getting details handled, not starting out. If you are asking how to start on Monday night, the TAs are directed to give a little direction and move on. We offer help every day except Saturday and most of those times are sparsely attended. Try going to one of those.

Office hours. By the way, no one comes to office hours. This has been true for some time. Towards the end of the semester I start to see people but the first half of the semester is almost unattended.

No one helped me with my problem! Neither the TAs nor myself are magicians. Student code is no more obvious to us than to you. You have to help us help you. Isolate the problem, simplify things, put cout statements in, something. Segfaults are particularly vexing. You have to run the debugger to at least find the line number where the segfault is occurring. By looking at your code we might know what is causing the problem, but if it takes ½ hour to dig it out the TAs have other people to help besides you.

Other Stuff

Prelabs suck! The pre-labs are designed to give you an experience in reading code and understanding it. In particular it helps you prepare for the exams since these are the kinds of questions you might get there. Since running prelabs, the exam scores have gone up slightly. It also gives you a different perspective (reading instead of writing code) on the topic of the week.

Labs suck! Our hope is that the material you cover in lab is the same material you need to do the projects. By working in a cooperative environment, you can work out some of the details of the topic of the week in lab, so you can better apply it. Sometimes the problems are hard, but this is the kind of problem-solving we are trying to get in the project. You don’t have to “finish” the lab, only make a good effort so it is an opportunity to try things out.

C++

Why C++. When you graduate from MSU, you will have experience with Python, a lovely language for general purpose work, and C++. There’s a quote from a modern C++ book I love:

“(If you’re not at all interested in performance, shouldn’t you be in the Python room down the hall?)”
— Scott Meyers, Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14
C++ gives you the opportunity to be efficient. An opportunity. Python might get you where you want to go, but with C++ you can get there efficiently. And that might matter quite a bit for the problem you are working on. In any event, having those two languages, Python and C++, gives you a very good perspective.

**C++ is much harder than Python.** Correct! C++ is a tough language to get a handle on. You should be taking it if you are a CS major or a major that needs C++. Otherwise, I think Python is the way to go. Modern C++, and we do teach C++ in a very modern way that you don’t often find in other places, is a better language than it was, but still difficult. But knowing how things work (like memory management) is important, even if you don’t end up using it all that much in your programming career.