How to Improve in CS106B

With the midterm graded and returned, many of you have asked for input about how to improve going forward. We're at the halfway point in the quarter, so in the grand scheme of things the night is young: there's plenty of time to solidify your skills, and even if you had a rocky start to the quarter you can end strong and be happy with the grade you receive.

The unifying theme of this handout is that you can get dramatically better outcomes from CS106B – or pretty much any class – if you're strategic and intentional with how you use your time.

Much of the advice in this handout is of the form "consider spending thirty minutes a day doing *X*, *Y*, and *Z*." I know that most of you are already putting a ton of time into this course and are making a good-faith effort to master the material. I don't want the takeaway from this handout to be "you need to put ***even more time*** into CS106B if you want to succeed." Quite the contrary. If you set aside some time every day to focus on high-leverage areas, I think you'll find that you spend ***much less*** time overall in CS106B, since each unit of time you're spending is significantly more productive. The question is how to spend your time more productively. Here are a few concrete suggestions.

Revisit your prior work and actively patch up any holes in your understanding.

In CS106B, it's very easy to structure all of your learning around trying to get through the current week's assignment. While this is a useful way to learn new concepts, if you're not intentional with how you approach the assignments you might not end up getting enough practice with the skills that you're weakest on.

After your IG on an assignment, pick a part of the assignment where you didn't do as well on as you would have liked. Read your SL's feedback and then take thirty minutes to rewriting your code to address that feedback. Take what you just wrote and either stop by office hours or the LaIR or post it on EdStem (privately!) to get the SLs to look over it. Specifically mention what issue you had on the assignment and see what the course staff thinks about your new version. If they think what you have is in good shape, great! You've internalized the feedback. If not, take this new piece of feedback and try rewriting things again.

This process will take some time – I figure you'll spend about half an hour a day working through this – but it's one of the single best things you can do to improve your performance. And from experience, this will dramatically decrease the amount of time you spend on the assignments. Chances are that in fixing things you'll find that you can tune the way that you approach problems in the first place, which will focus your efforts more effectively, or you'll find that you can write much less code than before, increasing your overall throughput.

Proactively ask questions about material from lectures.

It is extremely easy to get into a trap in CS106B where you fall behind on watching lectures, which means that you have to scramble to fit them in before the next assignment, which means that you fall behind on the next week's lectures, compounding the problem.

If you aren't regularly attending lectures (or watching them remotely on a fixed schedule), I would strongly recommend trying to get into the habit of doing so. We know this takes a lot of time, but it is absolutely worth it. Once you get into a good rhythm of staying on top of things, you'll have more time to pore over the material you're learning and think about how everything fits together.

If you are staying on top of lecture, I recommend spending thirty minutes a day actively reviewing what was covered. Here's one useful way to spend your time: pick a piece of code we wrote in lecture. Look back at your notes to see what the key insight or insights were in shaping that code. Then, without looking at the code from lecture, take a stab at rewriting that code from scratch. If you're having trouble doing so, that probably indicates that there's a concept that didn't click. Great! Review your notes again, pull up the slides, and if you still can't crack it, post a question on EdStem with what you've tried.

Manage to get the code working? Great! Compare what you have to what we wrote in lecture. Does it look similar? Is it

totally different? Your goal isn't to match what we have verbatim – that's pretty much never going to happen – but rather to see if you can reconstruct things based on your understanding of the ideas. If what you have looks really different than what we have, post it on EdStem or stop by office hours to ask for input. Maybe what you have is equally valid and just follows a different algorithmic strategy, in which case, fantastic! You just learned a new approach. Or maybe what you have doesn't work because of some minor nuance. In that case, even better! You just learned something new to watch out for, and probably got some feedback on your coding in the process.

Keep the SLs in the loop when studying.

As you're working studying, we recommend getting the course staff to help check your work. If you've worked through a practice problem (maybe one from section, or from the textbook, etc.), feel free to stop by the LaIR or office hours to get the SLs to review what you have, and take their input seriously! If you're getting feedback of the form "that's the right idea, but be careful about how you're writing it up," take another stab at writing up the answer and see what they have to say. If you're getting feedback of the form "there's a logic error here," ask the SLs how they arrived at that conclusion, make a note of their technique, and see if you can start applying that technique to your own code in the future.

Work in pairs, but do so strategically.

We encourage you to work in pairs on the assignments because if you're deliberate about the way you work in a group, you can learn much more than just flying solo. For example, if you code up your own solution to a problem and hand it to your partner for review, you can get their honest feedback about what they like, what they don't, and what parts they don't understand. The act of discussing your code in detail will then give you a much deeper understanding.

If you have been working with a partner, take a step back and reflect on your team dynamics. Are each of you doing all the problems co-equally (good), or is your partner always taking the lead (not so good)? Are you jointly working on the code (good), or is all the code your partner's (not so good)? Can you use your partner as a sounding board (good), or do they always give away the answers to the questions (not so good)? It can be easy in any relationship to arrive at a dynamic purely accidentally, but by being intentional about how you want your interactions to go, you can end up with significantly better outcomes.

Best of luck going forward!