Home About

CS 61A

Fall 2017

- **Lab**: 139B W 10:30-11:59 AM (Soda 277)
- **Discussion**: 139C F 10:30-11:59 AM (Soda 380)
- **Office Hours**: M/W 6:30-8 PM (Cory, check schedule)
- **Email**: kevinlin1@berkeley.edu

How to Learn Computer Science

High-level learning techniques for how to succeed in CS 1 courses like CS 61A, crammed into a 40-minute presentation for Golden Bear Orientation Fall 2017.

Most of the wordier advice I have to give has now been integrated with the course website as part of a big redesign this semester so everyone can access it more easily.

But since you're already here, let me share with you some additional insight into the research that interests me and motivates the work I do for CS 61A.

Metacognition & Mental Representations

Here's a selection of recent computing education research papers that provide insight into the learning processes that motivate how we learn CS.

- Computing Mentorship in a Software Boomtown: Relationships to Adolescent Interest and Beliefs (Ko, Davis)
- Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance (Loksa, Ko, et al.)
- Using Tracing and Sketching to Solve Programming Problems: Replicating and Extending an Analysis of What Students Draw (Cunningham, Blanchard, Guzdial)

One of the most useful skills acquired in the process of learning computer science is metacognition: an understanding of your own thought processes and, in particular, how much you know or don't know about a concept. Experienced programmers still make the same mistakes just like everyone else, and they still have to answer the question, "Why isn't it working?" just as often as everyone else.

What sets an experienced programmer apart is their ability to pull themselves away from the immediate frustration of a bug in the code and reconsider all the possible options. Experienced programmers excel at **metacognition** They've learned *how to ask the right questions* to themselves and adjust their solution bit-by-bit.

When faced with broken code, here are some of the questions I like to ask myself.

1. *Why* is this not working? Answering this question usually requires digging below the surface and looking at the context of the problem, so we might then ask,
2. *Where* is it not working *as expected*? There are two parts to this question that are important to answer.
    1. We reframe the goal of "why" in terms of "where" which allows us to look critically at specific portions of the code. This reduces the range of possibile changes down to a specific area and lets us ask even more specific questions about our code.
    2. In addition, we care why it is not working *as we expected it to*. One interesting observations that has been made about programming is that there are actually no such things as bugs or programming errors in the world, just **miscommunication**. Python executes code in a certain, predetermined pattern but the code we write and *our* understanding of Python is often incomplete which leads to perceived bugs. Let's try to rephrase the problem from solving bugs to reconciling with Python because, somewhere Python and I disagree on how to interpret the code. Where I see *this*, Python really sees it as *that*.
3. *What would happen* if I tried changing *this* value to *that*? Now that we've narrowed down where the problem must be, we can attempt to write a fix and reconcile with Python. The key to **learning** and overcoming this miscommunication in the future is not to edit code and make modifications blindly, but to *understand and envision Python's process*. Think of it as if you're arguing with a friend: empathize! Put yourself in their shoes! Think like Python!

The most effective and the most natural learning happens after we've walked through the path of **why**, **where**, and **what-if**. The key to becoming a great programmer is to learning how to *ask and answer increasingly-specific questions*.

Resources and Wisdom

- CS 61A Summer 2017 (my lectures are the ones marked *Video*)
    - Recursion
- Python 3 Documentation
- Online Python Tutor
- Owen's *Words of Wisdom (?)*
- Tammy's *Course Advice*
- James Maa's *A Beginner's Guide to Computer Science*
    - *Productivity Hacking Guide*
- Philip Guo on *How To Be Effective*
- Brian Harvey on *Why SICP matters*

Selected Staff Material

- Albert's website
- Allen's website
- Tammy's website
- Vivian's website