"Isn't it great when you propose something in class, you discover that the world goes exactly that way?"
— Anonymous student

Slightly paraphrased, this quote from a student evaluation captures my approach to teaching computer science: fostering the *joy of discovery learning*, grounded in *mutual trust* between instructor and students. My goal is to use joy and trust as a way to counter impostor syndrome, which is prevalent in CS higher education [7] and disproportionately affects students from underrepresented backgrounds [8]. I try to use every opportunity in my teaching to remind students: *You can do this. In fact, you just did!*

Philosophy 1: Joy Joy of learning takes many forms. In my classes, I guide students to experience the joy of discovery [3]: those moments when a student independently (re)discovers a result that is unexpected, elegant, or mathematically true. I believe this joy is a universal and powerful emotion. Compared to merely being fed knowledge, a first-hand experience of discovery enables students to practice key problem solving skills and contribute to a growth mindset [5].

To this end, I often *model* the discovery process in class by structuring lectures as guided explorations. Beginning with a simple, motivating problem, I invite students to try straightforward (if not naive) methods. Together we observe where these attempts fall short, and gradually refine them into a general algorithm. Throughout, I discuss and value *every* student proposal—even those that I know will fail—because the process of trial and error is more important than "one-shotting" the correct answer.

After class, this process continues through *inquiry- and discovery-based projects*. For example, in my undergraduate course on programming languages (initially as a TA) and graduate course on software verification (as a co-instructor), I developed projects where students gradually rediscover important concepts and algorithms. I begin with simple, concrete problems that build intuition for the general solution, then carefully remove scaffolds until students arrive at the anticipated idea themselves. I also intentionally *disguise* those hard problems in plain, simple clothing to remove the historical baggage and make them feel less daunting to approach. Only after students have overcome the challenge do I reveal to them the significance of their accomplishment: *"You could have been the first to discover this if you were born earlier!"*

Through these projects, my students have re-invented sequent calculus in proof theory[1], built a meta-circular interpreter for lambda calculus[2], and re-discovered non-chronological backtracking in modern SAT solvers[3]. Some students have reported that those projects were among the most rewarding parts of the course, and even reached out to inquire about related research opportunities!

Philosophy 2: Trust Genuine learning requires trial and error, yet I have observed that one of the biggest obstacles in computer science classes is students' discomfort in seeking help. This often stems from the perceived authority of the instructor and the fear of asking "stupid" questions in front of peers in an (artificially imposed) competitive environment [4]. So I strive to build a trusting relationship with each of my students and a non-competitive environment where they feel safe to express confusion and make mistakes.

In fact, I create an environment where mistakes are *encouraged*. For example, when I taught Programming Languages as an instructor, I experimented with a grading system where students earn "tokens" by asking or answering questions in class[4]. I emphasize *any* class-related contribution counts, even (and especially) questions that reveal confusion or misunderstanding. Students then redeem their tokens to retry

[1]Link to assignment : https://github.com/junrui-liu/CS162-TA/tree/winter-2024/homework/hw5
[2]Link to assignment : https://github.com/junrui-liu/CS162-TA/tree/winter-2024/homework/hw3
[3]Link to assignment : https://github.com/junrui-liu/CS292C/tree/master/projects/proj2
[4]A description of the token system in my syllabus: https://junrui-liu.github.io/cs162/syllabus.html#token-system

exam questions they got wrong, making assessments formative [2]. As one student put it, the token system was not only "helpful but motivates students to learn from mistakes."

I also cultivate a sense of perceived social support [6] early in the term through small but intentional gestures: learning every student's name by the end of the first week, sending surveys to understand their interests, and periodically checking in to adjust my teaching. As a concrete example, in my Programming Languages course, I noticed a student falling behind after the first quiz who had not attended office hours. I reached out and learned they had less CS background than their peers and were also balancing a part-time job. Together we created a personalized plan, including one-on-one review and problem sessions after their work hours, which helped them steadily catch up with the class.

Teaching Experience

As a teaching assistant I have served as a TA for eight quarters at UCSB and have received both departmental and college teaching awards for the past three years (see CV). In this role, I led weekly review and problem sessions for groups of 10–30 students, often incorporating *active learning* techniques such as think-pair-share and collaborative problem solving. For instance, in one compilers review session I designed a Jeopardy-style game[5], where two teams of students competed to answer customized prompts drawn from course materials as a playful way to review and earn extra credit.

As an instructor At UCSB, I taught CS 162, an undergraduate course on programming languages with 11 students[6]. Although I had extensive TA experience for this class, I chose to re-imagine it from scratch, incorporating the following elements: (1) I emphasized *learning by doing*. For example, I structured several lectures as workshops[7] where students used pencil and paper to *draw* the formal syntax and semantics of programs, helping them build physically grounded mental models. (2) I curated weekly *reflection assignments* that exposed students to broader implications of programming languages, such as feminism in language design[8].

I also co-taught CS 292C, Computer-Aided Reasoning for Software (Spring 2024, 17 students), a graduate course on software verification. In addition to developing a new module on interactive theorem proving and giving weekly lectures, I reworked the assessments, replacing them with three *research-based* assignments where students investigated and implemented key techniques in software verification.

Reflections and Growth

I love teaching, and this motivates me to keep improving. I actively seek feedback through mid-quarter surveys and informal check-ins, and I adapt my teaching based on what students share.

Outside of class, I make a point of keeping up with research in CS education and pedagogy. I have completed two teaching training programs at UCSB (Summer Teaching Associate Institute and Lead TA Institute) and am pursuing UCSB's Certificate in College and University Teaching. I also look forward to serving as my department's Lead TA this academic year, where I will train and mentor new computer science TAs in effective teaching practices.

These experiences have encouraged me to critically reflect on my teaching. One area I am working on is *student community building*. While I have focused on cultivating trust between instructor and student, my

[5] You can play my compilers Jeopardy game here (inspired by [1]): https://jeopardylabs.com/play/compiler-frontend-jeopardy
[6] Course website: https://junrui-liu.github.io/cs162/syllabus.html
[7] Example workshop materials: https://junrui-liu.github.io/cs162/lecture-notes/0701.html
[8] Reflection prompt: https://junrui-liu.github.io/cs162/reflections/week4.html

own experiences as a college student reminded me that peer relationships are equally important, and often longer lasting, for academic success and well-being. In future courses, I plan to revise projects to encourage collaboration and peer learning. I believe this is especially fruitful at a liberal arts college, where there is a consensus of valuing close collaboration, and I look forward to strengthening those ties and fostering new ones in my classes.

References

[1] Ruixia Bai. Game on: Engaging students with jeopardy in the classroom. In *Cultivating Student Engagement, Inclusion, and Success: UCSB Teaching and Learning Symposium*, Santa Barbara, CA, March 2025. Office of Teaching and Learning and UCSB Library. March 20–21, Library Room 1312.

[2] Benjamin S. Bloom. Learning for mastery. In Benjamin S. Bloom, J. Thomas Hastings, and George F. Madaus, editors, *Handbook on Formative and Summative Evaluation of Student Learning*, pages 43–57. McGraw-Hill, New York, 1971.

[3] Jérôme Seymour Bruner. The act of discovery. *Harvard Educational Review*, 31:21–32, 1961.

[4] Anael Kuperwajs Cohen, Alannah Oleson, and Amy J. Ko. Factors influencing the social help-seeking behavior of introductory programming students in a competitive university environment. *ACM Trans. Comput. Educ.*, 24(1), February 2024.

[5] Carol S. Dweck. *Mindset: The New Psychology of Success*. Random House, New York, 2006.

[6] Howard S. Friedman, editor. *Encyclopedia of Mental Health*. Academic Press, Oxford, 2nd edition, 2016.

[7] Adam Rosenstein, Aishma Raghu, and Leo Porter. Identifying the prevalence of the impostor phenomenon among computer science students. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, SIGCSE '20, page 30–36, New York, NY, USA, 2020. Association for Computing Machinery.

[8] Bianca Trinkenreich, Ricardo Britto, Marco A. Gerosa, and Igor Steinmacher. An empirical investigation on the challenges faced by women in the software industry: A case study. In *Proceedings of the 44th ACM/IEEE International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, pages 24–35, New York, NY, USA, 2022. IEEE / ACM.