Computers have become ubiquitous in society and the demand is ever increasing for people who can design software and understand the theory behind computational systems. These crucial skills open up nearly unlimited opportunities. I am a prime example of how these opportunities transform lives: **I am a first-generation Latino from a low-income migrant family who never previously believed I would get a Ph.D.** I want to diversify computer science and bring the same opportunities that I received to a wider segment of the public. I believe this is possible through effective teaching and standing out as a role model for others trying to follow the same path as I did. Ultimately, my experiences led me to approach teaching computer science with the following five principles:

1. Active learning and learning goals: Active learning has been repeatedly shown to increase student success [9]. All of my course material will facilitate student learning by having well-defined learning objectives that ensure my lessons are connected and build toward concrete goals. I incorporate live coding sessions when possible, where I periodically ask students for assistance and allow them to see me make mistakes. These sessions increase students' comfort when facing adversity while programming. Additionally, I ask low pressure questions during lessons that identify knowledge gaps and increase student engagement. For example, as a TA for Introduction to Programming II, I taught students how to create, modify, and traverse linked lists. I broke down the linked list into smaller learning objectives, first by creating a simple list of two nodes, then progressed to the more complicated cases of traversing and modifying. I used the whiteboard to visualize the processes that occur during each case to help the class conceptualize a linked list. When stepping through each case, I asked the class what the next step should be and how to update the linked list accordingly; I always made sure to praise students who answered and focused on the correct portions of any answer even if it was not completely accurate. After doing so, students felt more comfortable with the implementation of the linked list and left with a better understanding of this complex topic. In fact, I received multiple anonymous reviews stating "Very helpful to us and explained complicated C++ things".

2. Classroom inclusivity and empowerment: Students of all backgrounds should feel a sense of belonging in computer science, but this is hardly the case [8, 1, 10]. As an underrepresented scholar at a predominantly white institution, I experienced the lack of diversity within computer science and its impact. Historically, computer science has systematic issues with retaining students from underrepresented groups, while also failing to foster an environment where these students can thrive [6]. To address this issue, I cultivate a learning environment that emboldens future scholars and create a truly inclusive environment that nurtures students most at-risk of dropping out.

I try to reduce competition between students, as competitive classrooms harm all students' sense of belonging, especially first-generation students [2]. I incorporate collaborative learning within my teaching, which has been repeatedly shown to increase student success [9]. However, unfacilitated group work can lead to students from lower socioeconomic status to learn less than privileged partners [3]. I use "focusing questions" to address this disparity, where instructors guide students based on their current understanding and everyone contributes to the discussion [5]; this style of facilitation empowers students by letting them see themselves and their peers as learning resources. In Introduction to Programming II, I posed "focusing questions" during group work so that students could articulate and understand their thought process. This style of questioning allowed students to learn from one another and collaboratively accomplish the task at hand.

3. Practice makes progress: In computer science, we put theory into practice where problems can come from any discipline and be tailored for students of different backgrounds. Through my TA and outreach experiences, I generated different problems to explain the same concept for students that ranged from elementary to college. These experiences demonstrated that anyone can learn to program with the proper course material, guidance, and effort. To give students many opportunities to practice, I use programming projects with automatic software testing, while also providing additional programming challenges to help students develop a deeper understanding of the material. The use of automatic software testing will expose students to real-world software development practices, help them catch misconceptions early, and reduce grading time, allowing me to spend more time engaging with students directly [12]. In Introduction to Programming II, we used automatic software testing to provide students with on-demand feedback on the correctness of their programs. The software testing used unit tests to assess the correctness of individual functions within a project. This style of testing allowed me to emphasize the importance of breaking complex projects into smaller components, while also guiding students on how to do such breakdowns on their own.

4. Reading, writing, and reviewing code: Many computer science courses, especially early courses, focus on developing programming skills through individual programming assignments, which emphasizes the ability to *write* code. However, incorporating assignments where students must read, review, and present code increases student engagement and learning [7]. As such, I task students to inspect sample code to reinforce their mastery of programming and understand the thought process behind building a successful program. I also make students engage in critical thinking about code by reviewing samples with common mistakes. Pair programming also allows for reading, writing, and reviewing, where two students share one computer and alternate which one of them uses the keyboard. This style of programming emphasizes that the student without the keyboard read and review the code that is being written by the other student. In Introduction to Programming II, I paired students with similar programming expertise, which has been shown to increase student learning [4]. Reading, writing, and reviewing code are complementary skills that compound, allowing me to guide students to develop a well-rounded and highly effective programming repertoire.

5. Assessing fairly for mastery: As someone with no exposure to computer science prior to college, I understand everyone has different levels of knowledge when entering their first programming course. To overcome this barrier, I completed extra practice problems and attended frequent office hours to gain a better understanding of solutions. This effort gained me a mastery of course material, even though the extra work was not accounted for. Ultimately, this experience has influenced me to prefer mastery-based grading techniques like specification grading, as it reduces grading time and stress on the students, while improving student outcomes for middle to low performing students [11]. I will assign grades based on mastery of learning objectives and allow a limited number of assignments to be regraded, which lets students focus on mastering learning objectives instead of stressing over the number of points accumulated.

Conclusion: I look forward to continuing growing as a teacher, expanding my teaching repertoire, and developing courses for both majors and non-majors. I will solicit regular feedback by blocking off time for questions during each class period and using anonymous surveys. Even though the majority of my PhD was funded with fellowships and research assistantships, I actively sought teaching opportunities. I have helped lead multiple outreach activities that focused on teaching computer science to underrepresented students throughout Michigan, such as Girls Who Code and Technovation. These outreach experiences allowed me to teach computer science topics to students ranging from elementary to college and refine my explanation of computer science principles. Because of my interdisciplinary training, I am also well-equipped to develop courses introducing non-majors to coding, algorithmic thinking, and other computing concepts.

References

[1] Abeba Birhane and Olivia Guest. “Towards Decolonising Computational Sciences”. In: *Kvinder, Køn & Forskning* 29.1 (2021), pp. 60–73.

[2] Elizabeth A. Canning et al. “Feeling Like an Imposter: The Effect of Perceived Classroom Competition on the Daily Psychological Experiences of First-Generation College Students”. In: *Social Psychological and Personality Science* 11.5 (2020), pp. 647–657.

[3] Alexander Williams Chizhik. “Equity and status in group collaboration: Learning through explanations depends on task characteristics”. In: *Social Psychology of Education* 5.2 (2001), pp. 179–200.

[4] Brian Hanks et al. “Pair programming in education: a literature review”. In: *Computer Science Education* 21.2 (2011), pp. 135–173.

[5] Beth A. Herbel-Eisenmann and M. Lynn Breyfogle. “Questioning Our Patterns of Questioning”. In: *Mathematics Teaching in the Middle School* 10.9 (2005), pp. 484–489.

[6] Junming Huang et al. “Historical comparison of gender inequality in scientific careers across countries and disciplines”. In: *Proceedings of the National Academy of Sciences* 117.9 (2020), pp. 4609–4616.

[7] Christopher Hundhausen et al. “Integrating Pedagogical Code Reviews into a CS 1 Course: An Empirical Study”. In: *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*. Chattanooga, TN, USA: Association for Computing Machinery, 2009, pp. 291–295.

[8] Catherine Mooney and Brett A. Becker. “Sense of Belonging: The Intersectionality of Self-Identified Minority Status and Gender in Undergraduate Computer Science Students”. In: *United Kingdom &; Ireland Computing Education Research Conference*. Glasgow, United Kingdom: Association for Computing Machinery, 2020, pp. 24–30.

[9] Michael Prince. “Does active learning work? A review of the research”. In: *Journal of engineering education* 93.3 (2004), pp. 223–231.

[10] Yolanda A. Rankin, Jakita O. Thomas, and Sheena Erete. “Real Talk: Saturated Sites of Violence in CS Education”. In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 802–808.

[11] Kevin R. Sanft, Brian Drawert, and Adam Whitley. “Modified Specifications Grading in Computer Science: Preliminary Assessment and Experience across Five Undergraduate Courses”. In: *J. Comput. Sci. Coll.* 36.5 (2021), pp. 34–46.

[12] Chris Wilcox. “The Role of Automation in Undergraduate Computer Science Education”. In: *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. Kansas City, Missouri, USA: Association for Computing Machinery, 2015, pp. 90–95.