Computer programming and data science are pervasive examples of cognitively-complex computational tasks that tens of millions of people now perform. Writing code and analyzing data are critical for innovating in diverse fields, ranging from scientists making research discoveries to healthcare analysts devising public health policies to journalists compiling news stories using government datasets. However, due to the intricate and invisible nature of software, novices face immense cognitive barriers when trying to learn these skills. My research in human-computer interaction (HCI) combines theoretical foundations from cognitive science with scalable technical machinery from computer science to uncover such learning barriers and develop new ways to help novices overcome them.

Throughout my five years as an assistant professor, I took a three-pronged approach to investigating and mitigating some fundamental challenges of learning programming and data science: 1) performing empirical studies to uncover barriers faced by diverse learner populations, 2) designing visualization-based scaffolding to lower such barriers, and 3) developing novel tutorial formats that go beyond traditional text and video. My early faculty career research led to 31 publications mostly in top-tier HCI venues (see References section), a Best Paper award, four Honorable Mention paper awards, and an online education and research platform, Python Tutor [8], with over five million users from more than 180 countries.

Cognitive and Social Barriers Faced by Diverse Learner Populations: Most existing research on the challenges of learning programming have been done in formal school settings, most commonly in K-12 and university classrooms. However, nowadays far more people in more varied demographics are learning outside the classroom in informal settings such as professional workplaces and online communities.

My line of survey-based research with thousands of international participants uncovered unique learning barriers faced by diverse and previously-understudied populations. In the first known study of older adults learning programming [10] (CHI 2017 Honorable Mention award), 504 people from 52 countries reported age-related cognitive impairments (e.g., memory loss) along with feelings of social isolation as common barriers. Since programming languages and tools are English-based, I found non-native English speakers from 86 countries (N=840) struggling to deal with the dual cognitive challenges of translating their intentions into both English and computer concepts [11]. I also found that women programmers faced more social barriers than men (N=1470) when participating in the popular Stack Overflow online help community [6].

My colleague Parmit Chilana and I discovered a new technical population which we call conversational programmers: adults who learn coding to communicate better with their programmer colleagues, even though they do not need to write code on the job [3]. Conversational programmers are more diverse than their peers [4] (N=3151: e.g., more women, more humanities and social sciences majors) but are not served well by existing learning resources, which focus too much on the low-level logic of code rather than higher-level mental models of how software achieves user-facing goals [24] (CHI 2018 Honorable Mention award).

I am also amongst the first to study novices learning the rapidly-emerging fields of data science [18, 19] (CHI 2019 Honorable Mention award) and machine learning [1]. I found large cognitive disconnects between their aspirational high-level goals of producing data-driven insights and the low-level minutiae of configuring software environments, cleaning data, and tuning code parameters to work toward those goals.

More broadly, I have studied informal learning environments ranging from close-up peer mentoring at hackathons [27] to technology-mediated interactions in Massive Open Online Courses [12, 13, 17].

Visualization-Based Scaffolding for Learning Code and Data: Most people around the world do not have access to in-person tutors, so they must teach themselves using books and websites. To complement these static resources, I created dynamic visual scaffolding that supports both self-study and remote tutoring.

A fundamental challenge of programming is forming robust mental models of what happens step-by-step as the computer runs code, which is hard since this state is invisible. To help novices develop these mental models, I created Python Tutor [8], a website where people can write code (in languages such as Python, Java, JavaScript, C, and C++) and see automatically-generated visualizations of run-time state at each step. These visualizations mimic what a human tutor draws by hand. In the past six years, Python Tutor has grown to have over 10,000 daily active users and over five million total users from over 180 countries.
This large international user population provides a unique ability for me to experiment with novel tutoring interactions at scale. For instance, I augmented Python Tutor’s visualizations with a chat-based interface [14] to allow users to anonymously ask for help from others who are currently on the website. So far, tens of thousands of people have used this system to get tutored by complete strangers around the world. A chat log analysis showed technical knowledge exchange, impromptu social bonding, peer mentoring, emotional support, reciprocity, and prosocial pay-it-forward behaviors. This is also the largest naturalistic corpus of in-the-wild tutoring interaction data; it reveals a variety of novice misconceptions about programming.

Since tutor attention is especially scarce, I built an interactive dashboard visualization (Codeopticon [9]) that enables a single tutor to efficiently monitor dozens of learners and simultaneously help several at once while limiting cognitive load. I also developed new code visualization techniques [15], experience sampling methods for measuring learner frustration [5], experiments showing how learners from different countries have different code debugging habits [22], and text analyses showing how such visualizations can enhance discussion forums [31]. Python Tutor’s code is open-source, so nearly a dozen other research labs have built prototype systems and learning experiments on top of it, which further broadens its research impact.

Another major barrier for novice programmers and data scientists is installing, configuring, and managing the vast array of software tools that are irrelevant to the conceptual core of what they are trying to learn. To reduce this extraneous cognitive load, my work is amongst the first to use the web browser as instructional scaffolding: CodePilot teaches web development by integrating version control, software testing, and real-time collaborative coding into the web browser [26]. Fusion lets novices prototype web applications by extracting desired components from other webpages and hooking them together without needing to set up complex development tools [29]. DS.js [28] (UIST 2017 Honorable Mention award) and Mallard [30] augment the browser with data acquisition, manipulation, and visualization scaffolds that let novices learn basic data science and machine learning, respectively, using the data on webpages that they normally visit.

Interactive Tutorial Formats Beyond Text and Video: My empirical studies of novices learning programming from digital textbooks [25], online instructional videos [12], and discussion forums [31] have shown that traditional text and video formats are not expressive enough for conveying the dynamic intricacies of code and data. Thus, I developed novel interactive tutorial formats that improve upon text and video.

One unifying approach in this line of work is capturing tacit (unspoken) expert knowledge by observing user demonstrations and packaging it into a form that can be broadcast to many novices at once. Improv [2] allows tutorial creators to record live-coding demonstrations that are organized into interactive presentation slides to reduce cognitive load on both creators and viewers. Bespoke [23] and Torta [20] allow creators to make step-by-step software tutorials by directly running the required GUI and command-line applications and then annotating their workflows; user study participants found this demonstrational method of creating tutorials to be faster, more expressive, and less error-prone than manually writing them. Codemotion [16] uses computer vision to automatically extract code and data snippets from existing screencast tutorial videos and augments the video player so that learners can follow along by editing and running that code.

I also developed computational techniques to help people overcome expert blind spots (also called curse of knowledge), a cognitive bias where experts often skip over vital information when creating tutorials since they forgot what it was like to be a beginner who does not have the extensive prior knowledge that they do. Codepouri [7] crowdsources this work to Python Tutor users by letting them cooperatively annotate its runtime visualizations to make tutorials and to augment expert-created ones; in our study, their explanations often contained surprising details that experts did not think to include. Porta [21] (UIST 2018 Best Paper award) introduced tutorial profiling visualizations (inspired by software profiling) that reveal how a group of learners make their way through programming and data science tutorials; in our study, these visualizations showed tutorial creators precisely where learners struggled and how they could improve specific parts.

Conclusion: What makes my research portfolio unique is how it develops large-scale computational and data platforms and then uses them to: 1) make discoveries about diverse and previously-understudied international learner populations, 2) foster new kinds of instructional scaffolding and peer tutoring communities.
References


[17] Sean Kross and Philip J. Guo. Students, systems, and interactions: Synthesizing the first four years of Learning@Scale and charting the future. In Proceedings of the Fifth Annual ACM Conference on Learning at Scale, L@S ’18, pages 2:1–2:10, New York, NY, USA, 2018. ACM.


