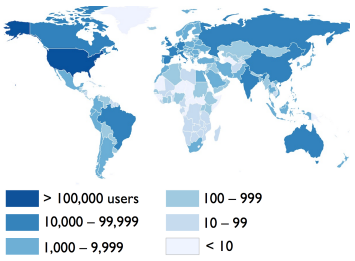
Philip Guo **Research Statement**

Millions of people around the world now want to learn computer programming and data science to work in fields such as software engineering, scientific research, business analytics, marketing, health informatics, public policy, and data-driven journalism. However, there are not nearly enough experts available to train everyone who wants to acquire these skills. To fulfill this need, ***I create interactive systems that help people learn computer programming and data science in a scalable way.***

I am a human-computer interaction researcher with a background in programming languages and software engineering. My research strives to make the current generation of programmers and data scientists more productive, and also to train the next generation in a scalable way. So far, I have built systems to make data scientists more productive in their data-centric programming workflow [10,11,12,13,14] and to bring the benefits of face-to-face tutoring to people who are learning programming online [1,2,3,9]. These systems have led to publications, funding, technology transfer to startups, and usage by ***over 1.5 million people in over 180 countries***. In the coming years, I plan to leverage this large user base to develop new systems that enable people to master all phases of the modern data science workflow, not just those involving programming.



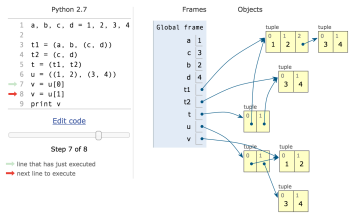
Over 1.5 million people in over 180 countries have used my Python Tutor code visualization system [1,2,3,9]. The map above shows the number of users per country. 25% of users are female. 7% are under 18 years old, 34% are 18–24, 43% are 25–54, and 16% are 55 or older. This large and diverse user base enables online experimentation at scale.

Interactive Systems for Learning Programming at Scale

Decades of computing education research have made headway on improving how programming is taught in K-12 and university classrooms. However, the vast majority of potential learners around the world *are not in a traditional school setting*, so they do not have access to in-person tutors who can watch them as they code, help them debug, explain concepts on-demand, and give encouragement to keep them motivated. To address this access gap, I created an online visualization and social learning platform [1,2,3,9] that provides some of the benefits of face-to-face tutoring.

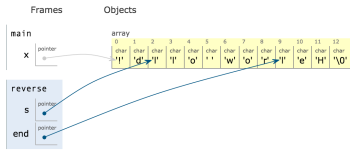
Python Tutor and the Rosetta Code Visualization Platform

One fundamental challenge of learning programming is forming a mental model of what happens step-by-step as the computer runs each line of code. Without this basic understanding, it is impossible to start becoming fluent in any programming language. However, this skill is hard for novices to develop since code execution state is invisible. To help learners form these crucial mental models, a tutor often draws diagrams containing variables, values, and pointers. But what if no tutor is available? To mimic how a human illustrates code execution, I created an automatic run-time visualization tool called Python Tutor (pythontutor.com) [9]. I then generalized Python Tutor's underlying visualization engine into a language-independent platform called Rosetta, which now visualizes code written in Python, Java, C, C++, Ruby, JavaScript, and TypeScript. Separate code instrumentation backends must be built for each language, but Rosetta provides an API that makes it easy to add new languages.



Python Tutor (pythontutor.com) enables the user to write code (left) and see a detailed step-by-step visualization of its run-time state (right). The diagram above shows a global frame, variables, nested heap data structures, and pointers.

Using a Rosetta-based visualizer such as pythontutor.com, a learner can write code in their Web browser, run that code on the server, and see an automatically-generated step-by-step visualization of its run-time state. These diagrams show the stack frames, variables, data structures, pointers, scopes, and closures at all execution steps. They allow learners to start forming proper mental models even when human tutors are not available. So far, 1.5 million people in over 180 countries have used Rosetta to visualize over 13 million pieces of code. Rosetta has been integrated into MOOCs from Coursera,

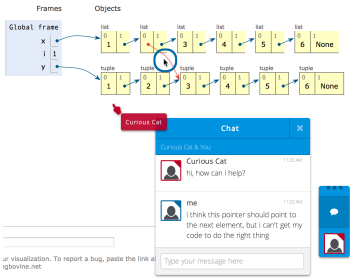


The Rosetta platform now visualizes seven languages: Python, Java, C, C++, Ruby, JavaScript, and TypeScript. This diagram shows a C function that uses two pointers to reverse a string on the heap.

edX, and Udacity and into the products of several education startups (e.g., Trinket.io, Zyante.com). This uniquely large user base enables me to deploy and evaluate new kinds of interactive systems at a scale that is not feasible in a lab setting.

Codechella: Multi-User Visualizations for Tutoring and Peer Learning

Rosetta works well for learning basics on one's own, but it cannot substitute for a human tutor. In a face-to-face setting such as a school computer lab, learners can summon tutors to sit with them to pair program, debug, and explain code execution. To bring these high-touch interactions to learners in online settings such as MOOCs, I built Codechella [2], a system that lets multiple people connect to a single Rosetta visualization session using a single URL. All participants can simultaneously edit and run code, explore visualizations together using multiple mouse cursors, and text chat. So far, in 17 months of live deployment on pythontutor.com, people from 475 cities across 60 countries have participated in 619 Codechella sessions to do both tutoring and peer learning. Most of these people could not have met face-to-face: 49% of sessions had participants from different cities, and 14% from different countries. Chat logs showed affective exchanges such as banter, camaraderie, and encouragement, which are reminiscent of face-to-face interactions, as well as signs of learning at the lower three levels of Bloom's taxonomy: remembering, understanding, and applying knowledge. Codechella is the first known multi-user run-time visualization system; it demonstrates that a shared real-time substrate for discussing code execution can mimic some of the intimacy of face-to-face tutoring for computer programming.



Codechella [2] enables multiple users to simultaneously write code, navigate the resulting Rosetta-generated visualizations using multiple cursors, and chat.

Codeopticon: Scaling Up Experts via Real-Time, One-To-Many Tutoring

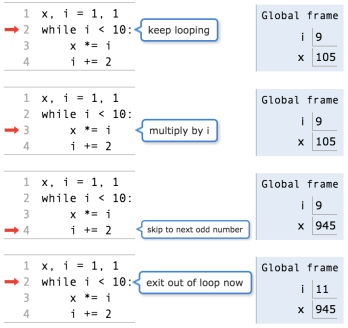
Codechella is effective for remote one-on-one tutoring, but there are usually far fewer available tutors than learners. For instance, in a large university course or MOOC, an instructor or TA may be responsible for helping hundreds or even thousands of learners. To make the most of experts' scarce time, I built Codeopticon [1], a one-to-many tutoring interface that enables a single tutor to monitor and chat with dozens of learners in real time. Each learner works in an online workspace containing a code editor, compiler, and Rosetta-based visualizer. The tutor sees a real-time view of all learners' activities on a grid of tiles, with each tile providing a concise summary of one particular learner's workspace. At a quick glance, the tutor can see how learners are editing and debugging their code, what errors they are encountering, and the full history of their code edits and visualization interactions. The interface automatically reshuffles tiles so that the most active learners are in the tutor's main field of view. The tutor can open an embedded chat box in any number of tiles to concurrently text chat with multiple learners who need help. A user study showed that 8 teaching assistants using Codeopticon for the first time successfully tutored anonymous learners from 54 countries who were working online on pythontutor.com. In a 30-minute session, each tutor monitored on average 226 learners, started 12 conversations, exchanged 47 chats, and helped 2.4 learners resolve their misconceptions. The tutors found their experience to be immersive, authentic, and at times even superior to face-to-face since they could use Codeopticon to see learners' edit histories, better control their own pace, and multitask. The design of Codeopticon points the way toward future interfaces for scaling up an expert's limited attention in real time in domains beyond programming.

Codeopticon [1] enables a single tutor to simultaneously monitor and chat with up to dozens of learners. Each learner's code editing and debugging activities are summarized in real-time in a grid of tiles.

Codepourri: Creating Coding Tutorials with a Volunteer Learner Crowd

Codeopticon makes the most of a single tutor's limited time and attention, but what if no tutor is even available? Since hundreds of learners are concurrently online on

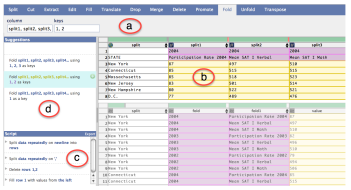
educational websites such as MOOCs or pythontutor.com, if a learner is trying to understand how a piece of code works, they can summon their fellow learners for help. To explore this idea, I created Codepourri [3], a crowdsourcing workflow where learners themselves create tutorials by annotating components within Rosetta-generated visualizations. In a one-week experiment, 101 learners on pythontutor.com used Codepourri to add 145 annotations to the visualizations of two pieces of Python code from an introductory-level textbook. Four Python experts judged 65% of those annotations as correct and 17% of the correct ones as containing surprising insights that even experts did not think to mention. The crowd then voted on the best annotations to include in a tutorial, and the experts judged the learner-created tutorials as comparable to those that experts would create. Codepourri advances our understanding of how to scale up the creation of programming tutorials by leveraging the collective time of a large crowd of online learners rather than using up the scarce time of a few experts.



Codepourri [3] enables a crowd to annotate execution steps within a Rosetta visualization (blue text bubbles shown above) and then vote on the best ones to create a step-by-step tutorial.

Data Science and Online Learning Research

For my Ph.D. dissertation in 2012, I was amongst the first to discover the unique programming challenges faced by the then-emerging population of data scientists. I built tools to make all phases of their data-centric programming workflow more productive: interactively synthesizing data wrangling (pre-processing) scripts [11], speeding up data analysis scripts via automated caching [12], making scripts more robust to malformed data [14], organizing data provenance and research notes [10], and creating portable code and data packages to make experiments reproducible [13]. Ideas from my data wrangling work [11] are now being commercialized by the startup company Trifacta, and my CDE packaging system [13] has been downloaded over 10,000 times and used at Google and NASA JPL. In the coming years, I plan to return to this line of work with a new online learning angle – to build tools that help people *learn* data science, not just to become more productive at doing data science.

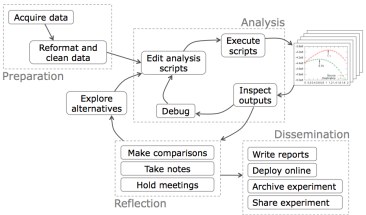


Proactive Wrangler [11] enables a user to interactively clean and reshape a data set in a spreadsheet-like interface. It makes proactive suggestions for what kinds of data transformations are most suitable for the user at each step.

As a postdoc at the online learning nonprofit edX and in Rob Miller's group at MIT CSAIL, I was amongst the first to conduct empirical studies of MOOCs: assessing the efficacy of different video formats [8] and how learner demographics affect course navigation strategies [7]. I also supervised Rob's Ph.D. students on their online learning projects: a data-driven educational video player [6], second language learning via instant messaging [5], and feedback amplification for algorithm design and coding assignments [4]. These projects have broadened my skills and interests in developing online learning technologies outside of my core focus on programming.

Research Agenda

While programming is a vital skill for software engineering, there is an order of magnitude more people across many professions who need to write code to work with data. Thus, my vision for the next decade is to build systems that enable people to master all phases of the modern data science workflow, even when expert tutors are not available. So far, I have made progress on systems for learning programming at scale [1,2,3,9], but writing code is only one component of data science. ***I plan to develop related systems for helping novices learn how to wrangle, assess, model, debug, and present data.*** I will leverage the unique size of Rosetta's user base (1.5 million total users, 5,000+ users per day) to deploy and evaluate my new systems. During my first year as an assistant professor (2014–2015), I have already received four grants to begin funding this vision: the NSF CRII (CISE Research Initiation Initiative award), a Google Faculty Research Award, a gift from Microsoft Research, and a University Research Award.



My Ph.D. dissertation identified four phases of modern data science workflows (preparation, analysis, reflection, and dissemination) and presented tools to help data scientists be more productive in each phase [10,11,12,13,14]. In future work, I plan to develop tools to help novices master all of these phases.

Specifically, I plan to explore questions such as:

- How can people learn to effectively use multiple languages together in their work (e.g., a mix of C++, Fortran, Python, R, Bash, and Make for scientific research)? While there is plenty of support for homogeneous single-language workflows, there is almost no tool support for learning how to productively work with multiple languages at once. Rosetta is the only known multilingual run-time visualizer, so I can build tools upon it to enable learners to form robust cross-language mental models.
- How can interactive visual tutorials help novices develop intuitions about data quality, numeracy, statistics, machine learning, and experimental design?
- Can intelligent interactive systems help novice data scientists avoid common experimenter biases, statistical misconceptions, and erroneous data interpretations?
- Can we generate more compelling personalized data analysis tutorials (as opposed to generic text or video tutorials) by mining one's own data such as emails, fitness trackers, or computer activity logs?
- How can crowd-powered critique systems train data scientists to become better technical writers so that they can more effectively communicate the results of their analyses?

Through my research, I strive to innovate in ways that both advance the academic field and also impact millions of people, as I have done so far with pythontutor.com and the Rosetta visualization platform. I am passionate about working toward a world where everyone – not just those with access to formal schooling – can fulfill their potential to master computer programming and data science. These are two crucial skillsets for finding enriching and sustainable careers in the coming decades as computation and data pervade even more facets of modern life.

References

1. **Philip J. Guo**. Codeopticon: Real-Time, One-To-Many Human Tutoring for Computer Programming. UIST: Symposium on User Interface Software and Technology, 2015.
2. **Philip J. Guo**, Jeffery White, Renan Zanelatto. Codechella: Multi-User Program Visualizations for Real-Time Tutoring and Collaborative Learning. VL/HCC: Symposium on Visual Languages and Human-Centric Computing, 2015.
3. Mitchell Gordon and **Philip J. Guo**. Codepourri: Creating Visual Coding Tutorials Using A Volunteer Crowd Of Learners. VL/HCC: Symposium on Visual Languages and Human-Centric Computing, 2015.
4. Elena L. Glassman, Jeremy Scott, Rishabh Singh, **Philip J. Guo**, Robert C. Miller. OverCode: Visualizing Variation in Student Solutions to Programming Problems at Scale. TOCHI: Transactions on Computer-Human Interaction, 2015.
5. Carrie J. Cai, **Philip J. Guo**, James Glass, Robert C. Miller. Wait-Learning: Leveraging Wait Time for Second Language Education. CHI: Conference on Human Factors in Computing Systems, 2015.
6. Juho Kim, **Philip J. Guo**, Carrie J. Cai, Shang-Wen Li, Krzysztof Z. Gajos, Robert C. Miller. Data-Driven Interaction Techniques for Improving Navigation of Educational Videos. UIST: Symposium on User Interface Software and Technology, 2014.
7. **Philip J. Guo** and Katharina Reinecke. Demographic Differences in How Students Navigate Through MOOCs. L@S: Conference on Learning at Scale, 2014.
8. **Philip J. Guo**, Juho Kim, Rob Rubin. How Video Production Affects Student Engagement: An Empirical Study of MOOC Videos. L@S: Conference on Learning at Scale, 2014.
9. **Philip J. Guo**. Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education. SIGCSE: Technical Symposium on Computer Science Education, 2013.
10. **Philip J. Guo** and Margo Seltzer. Burrito: Wrapping Your Lab Notebook in Computational Infrastructure. TaPP: USENIX Workshop on the Theory and Practice of Provenance, 2012.
11. **Philip J. Guo**, Sean Kandel, Joseph M. Hellerstein, Jeffrey Heer. Proactive Wrangling: Mixed-Initiative End-User Programming of Data Transformation Scripts. UIST: Symposium on User Interface Software and Technology, 2011.
12. **Philip J. Guo** and Dawson Engler. Using Automatic Persistent Memoization to Facilitate Data Analysis Scripting. ISSTA: International Symposium on Software Testing and Analysis, 2011.
13. **Philip J. Guo** and Dawson Engler. CDE: Using System Call Interposition to Automatically Create Portable Software Packages. USENIX Annual Technical Conference, 2011. (short paper)
14. **Philip J. Guo**. Sloppy Python: Using Dynamic Analysis to Automatically Add Error Tolerance to Ad-Hoc Data Processing Scripts. WODA: International Workshop on Dynamic Analysis, 2011.