Motivation Though more students than ever are graduating with Computer Science (CS) degrees [1], a major gap still exists between the skills of these graduates and the skills needed for industry success [2]. Two major components of this gap are programming skills (e.g. code style) and computational thinking (e.g. extending algorithms). ***My research will enable teachers to close these gaps by building a toolkit to integrate better-targeted problem types than those currently in use.***

Programming assignments in CS classrooms, primarily code tracing and code writing, lack the granularity to target students' Zones of Proximal Development (ZPD, i.e. challenging but solvable problems). Code tracing questions involve reading and understanding code, but they may not trigger students' mental models of concepts because they are tedious or too easy [3, 4]. Code writing questions are a large leap from code tracing, conflating many programming skills into a single solution (e.g. design, computational thinking, programming, code style) and supporting a wide space of disparate approaches and solutions.

Extrapolating from earlier results [3], I propose the use of ***Parsons problems*** (e.g. [5]) to help students remain in their ZPD while ***learning computational thinking and programming through advanced CS concepts***. Parsons problems involve unscrambling chunks of code, often individual lines, from a target program into a correct solution. Such problems provide similar learning gains to code writing problems – ***often 30% faster*** – for introductory assignments [3]. ***My research will enable teachers to integrate Parsons problems into existing curricula to better attain their teaching outcomes by providing a toolkit for these new problems***.

Preliminary Work Within my Ph.D., I have been studying how Parsons problems can be applied to improve CS pedagogy. In Spring 2018, I ran an exploratory between-subjects study which found that when students begin by solving a Parsons problem instead of writing code, they are ***more able to generate multiple alternative solutions*****,** addressing a skill gap in the "ability to generate alternate solutions" [2]. In Fall 2018, I ran a within-subjects study, followed by a structured interview, teaching algorithms to students who had not taken an algorithms class. They were taught two algorithms by either writing code from a pseudocode specification (i.e. a language-independent solution) or by solving a Parsons problem. Students with a range of skills noted that Parsons problems let them focus on the logic of the algorithm, engaging them in computational thinking. Students found writing code from pseudocode to be either too complex, distracting their focus, or too trivial, not engaging with the algorithm. These results suggest that Parsons problems can support ***a variety of students in practicing their computational thinking***.

Approach In my thesis research, I will explore how we can leverage new problem types, e.g. Parsons problems, to supplement existing teaching tools for advanced CS concepts. I will run longitudinal studies on these new problem types by partnering with UC Berkeley professors who teach relevant classes with hundreds of students. I will then synthesize the results from these studies to develop a toolkit for teachers to help teachers easily adapt existing material into Parson problems and achieve their curricular desires.

RQ1: How can Parsons problems support the development of computational thinking with algorithms? Based on my Fall 2018 study, I am now exploring how to help students further focus on computational thinking with new Parsons problems that flexibly blend pseudocode with code within or between problems. By using pseudocode, students are constrained to rely less on syntax and more on computational thinking. I will measure students' learning gains of the taught algorithms as well as their performance on interview-style algorithm questions.

RQ2: How can Parsons problems improve programming ability by teaching programming idioms? One major risk of supplementing existing assessments with Parsons problems is that it could hurt students' programming skills by reducing their time spent practicing writing code. To overcome this pitfall, I will explore how Parsons problems can improve code writing by teaching programming idioms (i.e. common code and design patterns such as finding the five largest numbers in a list). There is a strong connection between students' ability to write well-styled code and their ability to select and apply appropriate programming idioms [6]. However, complex idioms are often not explicitly taught. Results from my Spring 2018 study indicate that Parsons problems could support students exploring multiple solutions to a problem using different idioms, helping them compare when they are effective to use. They could also expose students to a range of problems where an idiom is applicable, helping students learn how it can be applied. I will run a formative study to better understand how students learn and select idioms. I will measure ABC scores of solutions, a well-established metric for code complexity, to evaluate students' ability to efficiently apply idioms [6].

RQ3: How can instructors easily integrate Parsons problems into their teaching? Even if these new problem types are found an effective teaching tool, it must also be straightforward for instructors to generate and integrate them into the classroom. I will interview professors to explore the range of teaching methods used in classes and homework. These interviews will guide the design of a system to give teachers a powerful tool to target specific learning goals with more problem types. For example, a teacher could use think-pair-share in class to encourage students to share their problem-solving strategies by having students discuss which line of pseudocode should be placed next in a Parsons problem. Or, the large corpora of student solutions could automatically generate Parsons problems for teachers to modify and use.

Resources UC Berkeley provides rare access to collaborate with teaching professors in large, innovative classrooms. Here, my research will ***change how thousands of students learn CS***.

Intellectual Merit My work will enable further research of teaching tools throughout the CS curriculum. My proposal explores how students learn and use computational thinking and programming idioms in complex problems through problem types and assessments. While there is a plethora of research on teaching introductory CS concepts, there is minimal research on how to teach advanced CS concepts, which are closer to real-world needs. The results of my work will ***empower teachers to create new resources to help students learn complex CS concepts.***

This research will be the first to explore the effectiveness of Parsons problems beyond introductory courses, ***creating new interactions and contexts for Parsons problems***. This will inspire applying Parsons problems in new ways: incorporating them into more domains in CS curricula, using them for post-school learning such as API tutorials or system documentation, or new situations where engaging with multiple solutions is beneficial.

Broader Impact My research is inspired by a desire to make CS concepts ***more accessible***. Code writing problems are "one of the most significant reasons for giving up" by online learners in introductory classes [3]. These techniques will help ***improve learners' self-confidence*** in these areas, with the aim of ***reducing impostor syndrome and attrition***, as a step towards making ***CS programs more inclusive***.

The results of this research will be disseminated within top-tier publications in HCI and CS Education. My software engineering experience enables me to make the toolkits developed over the course of my research robust and publicly available. Together, these will help researchers and content creators ***make knowledge more accessible to a diversity of audiences.***

[1] https://nces.ed.gov/programs/digest/d17/tables/dt17_322.10.asp **[2]** Radermacher et al. "Gaps between industry expectations and the abilities of graduates," SIGCSE '13 **[3]** Ericson et al. "Solving parsons problems versus fixing and writing code," Koli Calling '17 **[4]** Denny et al. "Evaluating a new exam question," ICER '08 **[5]** https://js-parsons.github.io/ **[6]** Wiese et al. "Teaching Students to Recognize and Implement Good Coding Style," L@S '17