Analyzing interviews on computational thinking for introductory physics students: Toward a generalized assessment

Justin Gambrell and Eric Brewe

Department of Physics, Drexel University,
Disque Hall, 32 S 32nd Street, Philadelphia, Pennsylvania 19104, USA

(Received 7 August 2023; accepted 5 March 2024; published 26 April 2024)

Computational thinking in physics has many different forms, definitions, and implementations depending on the level of physics or the institution it is presented in. To better integrate computational thinking in introductory physics, we need to understand what physicists find important about computational thinking in introductory physics. We present a qualitative analysis of 26 interviews asking academic ($Na = 18$) and industrial ($Ni = 8$) physicists about the teaching and learning of computational thinking in introductory physics courses. These interviews are part of a long-term project toward developing an assessment protocol for computational thinking in introductory physics. We find that academic and industrial physicists value students' ability to read code and that PYTHON (or VPYTHON) and spreadsheets were the preferred computational language or environment used. Additionally, the interviewees mentioned that identifying the core physics concepts within a program, explaining code to others, and good program hygiene (i.e., commenting and using meaningful variable names) are important skills for introductory students to acquire. We also find that while a handful of interviewees note that the experience and skills gained from computation are quite useful for student's future careers, they also describe multiple limiting factors of teaching computation in introductory physics, such as curricular overhaul, not having "space" for computation', and student rejection. The interviews show that while adding computational thinking to physics students' repertoire is important, the importance really comes from using computational thinking to learn and understand physics better. This informs us that the assessment we develop should only include the basics of computational thinking needed to assess introductory physics knowledge.

DOI: 10.1103/PhysRevPhysEducRes.20.010128

I. INTRODUCTION

Computational thinking (CT) has been a growing area of interest due to its applicability across many disciplines [1,2], and the Next Generation Science Standards (NGSS) placed CT as a key practice of science and engineering in 2013 [3]. Within physics in particular, there is a synergy between learning discipline-specific content and developing CT skills: physics bolsters CT skills and practices by providing a context rooted in mathematics while CT practice bolsters physics learning by streamlining the process and expanding the physics that is accessible [4–9]. Because of this synergy, incorporating CT into the physics degree is a logical outcome. However, curricula in introductory physics often limit their content to problems that can be solved analytically. To better serve physics education, therefore, we need to examine how we can update and modernize these curricula to meet the challenges of incorporating CT and maintaining the relevance of physics in the preparation of all students [10,11].

Multiple researchers and curriculum designers have attempted such reforms by implementing and evaluating CT in physics and other science, technology, engineering, and mathematics (STEM) curricula. Projects and Practices in Physics (P^3), for example, is a curriculum at Michigan State University (MSU) in which students solve highly contextualized problems computationally within small groups. Researchers have examined group interactions in this context [12,13], finding that the ways the group interacts with finding bugs in a program are either more strategic by checking line by line or less strategic by playing (a form of guess and check) with the program until the bug was found. In a phenomenographical study of (P^3), Hawkins *et al.* found that in their theme of "computation helps to learn physics," a majority of the students interviewed mentioned variations that pointed toward students having some positive computational experience in the class [14]. Hawkins results are promising as computation in conjunction with physics can often be viewed negatively by students [15]. Finding implementations that can reduce computational fear for students is important for further

adoption. The Partnership for Integration of Computation into Undergraduate Physics (PICUP) is a separate organization that focuses on providing resources for instructors wishing to add computation into their undergraduate physics curricula [16]. Additionally, the American Association of Physics Teachers Undergraduate Curriculum Task Force has released a set of recommendations for computational physics in the undergraduate physics curriculum. Among others, these recommendations include that physics departments should be integrating computation into the curriculum. However, the authors also identified the lack of research on the assessment of CT skills as a problem [10].

Thus, the integration of computation into the introductory physics curriculum requires the development of assessment protocols for CT that can be used across a variety of classes, but this tool does not yet exist. Our research is working toward the development of such an assessment. Developing assessments is an iterative and cyclical process in which one identifies the criteria to be assessed, gives and evaluates the assessment, and reflects upon the validity of the assessment [17,18]. In this paper, we address the first of these three stages by identifying the learning objectives of a computational introductory physics course. To do so, we conducted interviews with 26 academic and industrial physicists to identify broad learning goals in a computationally integrated introductory physics class, which will guide our assessment development in the future. We specifically aim to answer the following research question:

- What are the learning goals for a computationally integrated introductory physics class?

A. What is computational thinking?

CT is a complicated concept regardless of your experience level with it. Thinking and consequently CT is esoteric. A definition that varies based on interpretation can be confusing when it comes to scientific research. One such way to address this issue in definition perspective is to provide and explain our own definition which has parts from other definitions provided by Wing and Weintrop:

In general, CT is a specialized form of thinking involving problem decomposition, pattern identification or recognition, abstraction of complicated systems along with proper justification, and algorithms brought to bear in order to solve problems that necessitate numeric modeling and problem solving (often involving the use of computers to aid the solving of these problems) [1,19]. Computational literacy is the state of having competent knowledge and use of computational programming environments. Computational literacy is not needed to perform computational thinking in physics; yet, to manipulate physics programs, some computational literacy is required. Computational thinking in physics is thinking in such a way that computations describing physical phenomena are streamlined, efficient [20], and well documented, and this is typically executed via programming environment.

B. CT in introductory physics

CT is an emerging fundamental skill. Only in the past 70 years has it become a specific focus in education [21]. In particular, Wing in 2006 popularized the idea of CT and has in part fueled the integration of CT in many disciplines [1,22]. Providing physics students the opportunity to learn CT skills will not only help them obtain jobs in their future but also help develop and promote more ideas that may have been left behind otherwise.

In PER, computational thinking research is sparse compared to other areas. A paper in 2020 by Odden demonstrates this. In his paper, he thematically coded 18 years (2001–2018) of physics education research conference proceedings using natural language processing. Of the ten themes found, none of them involved CT or CT education [23]. There is decent literature involving general computational thinking research, but it drops off as we get more specific to STEM and physics. This sparseness presents an excellent opportunity for researchers to explore the various facets of physics computational thinking education.

A report from the AIP Statistical Research Center investigated the initial employment of physics bachelors and Ph.Ds in 2019 and 2020. According to the report, 46% of new physics bachelors are employed and 59% of these are employed in the private sector. Of those 59%, 35% work in an engineering field and 24% in a computer science field. Over 50% of these engineers and computer scientists indicated that they use programming on a daily basis [24]. Computational practices are expected for some of our physics graduates, and so we should be equipping them with skills they might need along the path to their degree.

Introductory physics is in many ways an ideal space to begin learning computational methods while learning physics. In introductory physics classes, standard textbook problems are a primary means of assessment of students [25]. These problems are typically limited to those that can be solved analytically. Students are given idealized problems that often have internal assumptions made for them. While the reason for introducing problems in this way is to help better understand physics with basic examples, it has been inadvertently stifling physics education [26]. Students have the means and the access to cheat the current educational system. For example, students can use Chegg to answer book problems immediately. This has the potential to perpetuate inequality in the classroom, as a student could essentially purchase a good grade. Chegg does not have computational method solutions as of yet, although ChatGPT's introduction and recent popularity may even complicate computational assignments. A recent study by Kortemeyer found that in having ChatGPT take an introductory physics course, ChatGPT outperformed the other real students on the programming assignments [27]. ChatGPT is a language model, so it definitely has its limitations when doing physics, but this highlights what can come in terms of artificial intelligence. More work will

need to be done to learn how we can leverage services like ChatGPT to supplement student learning.

Computational methods might force students to make their own assumptions explicit and foster creativity. Odden and Caballero produced a paper on a pilot study on computational essays. One of their results was that students reported that the computational essays helped facilitate creative investigation [28]. Computational methods allow exploration of different physics principles and can address this early on in their physics career so that they may continue to build creativity and truth of messiness in physics throughout their education.

Explicit integration of computational methods and thinking in introductory physics courses can present challenges. For example, it may be that on top of current math prerequisites or corequisites, these courses might need computational prerequisites or corequisites. This is likely an institutional level decision and cannot be determined solely by the physics community. That being said, we do argue that CT should be explicitly added to the introductory physics courses for physics majors. Classes for physics majors typically have a smaller number of students. This means that instructors have more opportunities to provide individual support to them all, and learning these CT ideas potentially prepares students for their future careers [24].

II. LITERATURE REVIEW

A. Computational thinking in STEM

There is an ongoing effort to integrate CT into many aspects of STEM curricula and degree programs. Samar Swaid examined the many aspects of CT and how they might be integrated into different STEM courses. Swaid took careful note of what affordances CT could bring to each individual STEM discipline. For example, calculus I and II afforded a great environment for abstraction, data, and retrieving, but not so much for algorithms, design, and evaluation. Alternatively, biology I and II afforded all of those aspects besides abstraction and design [29]. While the integration of CT into STEM is generally agreed upon as the correct course of action, it is important to note that CT is going to look different for each discipline. This paper serves as a way to further define what CT looks like in the physics context.

A paper by Harangus and Kátai presented a question to be solved using algorithms for secondary and higher education students and found a relation between reading comprehension and problem-solving skill. They noted that many of the students did not even attempt the problem if they viewed it as too difficult [30]. This work is important because it points out the motivation of students when tackling CT-intensive problems. Physics, programming, and STEM overall are commonly associated with being high intelligence fields. This conception can be especially detrimental toward the attitudes of students taking STEM courses. In order to help retain students in STEM, the misconception of it being only for the intelligent students needs to be addressed. It is very important for STEM fields integrating CT to consider the attitudes of their students. CT problems should be introduced in such a way that eases students in. There should be a clear scaffolding of these CT concepts and practices so that students do not feel that the problems they encounter are too hard to even attempt.

In a literature review of CT in STEM, Wang *et al.* found that few studies examined equity associated with these courses [31]. Most of the studies that addressed equity, did so focusing on various student groups like gender, socioeconomic status, and geographic location. There was one paper that focused on instructional strategies to increase equity. Wang also found that while studies assessed student outcomes with CT integrated into the course, there were limited studies of assessments of the integrated CT course itself. In this paper, we do not directly address the equity of CT in introductory physics. We recognized that CT can be intimidating to students at first but did not investigate how it may differ for different student populations. Our focus of this paper is on the learning goals of a CT-integrated introductory physics class. We chose not to focus on equity because we did not want to do too many things at once and divide our attention. Equity in CT in physics is important and needs to be researched carefully and wholeheartedly.

Malyn-Smith and Lee developed the "learning occupation" of a CT-enabled STEM worker and then identified and validated it with expert CT workers the computational thinking skills or competencies that are used by scientists and engineers in STEM careers [32]. In their work, they found eight job functions that a computational thinking enabled STEM professional would use. They were identifies problem, specifies constraints, designs the model or system, builds the model, develops experimental design, verifies the model, optimizes the model and user-interface, and facilitates knowledge or discovery. They provided examples of CT in action for students. They also showed educational programs that foster CT development bases on specific tasks within the job functions. Some programs covered more tasks but were also at a higher educational level. This work helps to further identify what we as educators are looking for in CT skills and how exactly we can foster or assess this development.

Lyon and Magana examined model-building activities for engineers to elicit CT [33]. Besides decomposition, algorithms, and abstraction, they also added evaluation and generalization to their CT practices. They found in their study that there were more outcomes in the evaluation practice than in the others. The outcomes included time efficiency, code flexibility, code accuracy, design criteria, code complexity, debugging, and solution usability. They suggested that model building activities can help students build their CT skills and that some skills are more focused than others.

Weintrop *et al.* in 2016 defined computational thinking for mathematics and science classrooms. They said that with more traction of making CT a core scientific practice, a concrete definition of CT is needed along with a theoretical grounding for the form it should take in classrooms. They provided a definition of CT with four main categories: data practices, modeling and simulation practices, computational problem-solving practices, and systems thinking practices. There are 5–7 subcategories within each. They did this by reading CT literature, interviewing mathematicians and scientists (mostly physicists and physics instructors), and reading good CT instruction materials. They emphasized that the definition is CT in the science and math context. They created three lessons (physics, biology, chemistry) to implement their taxonomy and stated that implementing it in high school reaches a wider audience [19]. Weintrop defined the computational thinking practices of a science course and provided examples of how they implement these practices in a physics class. In designing our assessment, we will use this framework to connect physics learning goals to a specific CT practice.

There are a few other works where computational thinking and physics were examined with Weintrop's Taxonomy in mind. The Bootstrap for physics initiative for high school physics teachers is another area where Weintrop's framework is used [34]. This was proposed as a way to provide professional development for high school teachers so that they could teach computational practices to their students. They used modeling instruction pedagogy as it implicitly emphasizes many computational thinking skills. Vieyra *et al.*, in an interview analysis of 12 physics teachers and their views on computational modeling, incorporated Weintrop's framework [35]. They had their participants discuss as a group about the taxonomy that Weintrop presented. They used this in conjunction with bootstrap as a way to address a lack of consensus on the definition of CT. Orban and Teeling-Smith examined CT in introductory physics from the lends of Weintrop's framework. They defined CT and described how computational thinking aligned with physics instruction [36]. They also discussed how the modeling instruction pedagogy mixed well with CT instruction as modeling instruction emphasizes multiple representations, and a computer program provides another representation.

We see from these papers on CT in STEM that there are many aspects to integrating CT into a STEM course. Weintrop focused on the practices that students should learn to develop their CT skills. Wang reviewed and discussed the equity of CT in STEM finding that there were very few studies on this intersection. Although this is currently an understudied area, the importance of both CT and equity will push research in this direction when CT in specific disciplines is better understood. Harangus found the relation between reading comprehension and problem-solving skill. They also pointed out that problems that were seen as too difficult by the students were not even attempted. This can be an issue in physics. Finally, Swaid discussed how different STEM courses afford different CT aspects. With a general foundation of knowledge of CT in STEM, we can narrow down our focus to how these aspects are present in the physics domain.

B. Computational thinking in physics

Caballero *et al.* in 2012 implemented and assessed computational modeling in introductory mechanics. They claimed that implementing computation in intro physics courses had many benefits: modeling processes make complex problems tractable, and computation can explore the applicability and utility of physical principles. They also affirmed that students who compute are doing work that is more representative of their potential future work as scientists and that learning how to debug code is part of learning computational modeling. They questioned the challenges students face when learning and applying computational problem-solving techniques, and how they could mitigate those challenges through instruction. They found no statistical difference between the performance of analytic vs computational homework. They also found that students with some previous programming were no more successful than those without. They introduced computation to a large enrollment calculus-based mechanics course at the Georgia Institute of Technology. Students were taught the computational language of VPYTHON. [37]. The first thing we discuss in this paper is the two benefits of tractable complex problems and physical principal exploration. Introductory physics education as we have said before is often limited in what can be solved analytically. It is not that it cannot be done or is too difficult but that there is not enough time in a semester to spend the time effectively exploring these phenomena analytically. Computational modeling provides the tools to tackle complex problems in a feasible and time-efficient way. Along these lines, it also provides a place to play and explore. Computational modeling allows quick changes of parameters so that students can explore more nuances of physical phenomena. Again, it is not that this cannot be done analytically, it is just that computational modeling provides a feasible and time-efficient way to learn and explore physics.

Obsniuk *et al.* in 2015 did a case study on novel group interactions through introductory computational physics. They questioned what computation of physics in a group setting looks like. Their research was on an extended version of M&I (Matter and Interactions), where students work with computational physics in a group setting. M&I was developed by Chabay and Sherwood and is a VPYTHON based curriculum. Students working with computational physics in a group setting was a part of Projects and Practices in Physics or (P^3). They found two distinct strategies suited to computational tasks. They focused on the social exchanges between group members and the interactions between the group and the computer. The

group and computer interactions varied from actively sifting through code to observing a 3D display. Students were to debug fundamentally correct code with wrong physical results. Bug recognition and bug resolution were the two necessary limits on physics debugging. When students found a bug, they blamed the error on their understanding of PYTHON and not their understanding of physics. Students parsed through every line of code, asked each other if the line was correct, and moved on. The paper classified two debugging types as more strategic and less strategic. Self-consistency was when they checked line by line and confirmed with each other (more strategic). Play or productive messing about was the less strategic way. Play showed the benefit of immediate visual display as a check where analytic tasks did not have that [12]. This paper examined the intersection of computational physics and group work interactions. Researching how students interact with each other and the programming environment are important because teamwork and adjacently communication are skills used on a daily basis for physics graduates employed in the private sector [24].

Weller *et al.* introduced a CT framework. They described 14 practices that emerged from an integrated computational physics course: decomposing, highlighting and foregrounding, translating physics into code, algorithm building, applying conditional logic, utilizing generalization, adding complexity to a model, choosing data representation form, intentionally generating data, analyzing data, manipulating data, debugging, demonstrating constructive dispositions toward computation, and working in groups on computational models [38]. Many of these practices that emerged in a computational physics course were practices that were found in Weintrop's paper. There are a few that are not directly matched. For example, working in groups on computational models and applying conditional logic are not found in Weintrop's CT practices. This may be an example where conditional logic is better suited in the physics context than the general STEM context.

CT practices are getting better defined for physics in general, but there is still a lot of research to be done in many of the specific physics courses. The CT practices of an introductory course will be different from a capstone course. It may be that the practices are the same, however the sophistication of the practice would be different, which is still an important distinction. We will now discuss the methods of our project and how we went about determining learning goals for introductory computationally integrated physics course.

III. METHODS

A. Participant selection

To identify learning objectives in a computationally integrated introductory physics course, we conducted interviews with participants having some relation to the field, either within academia or within industry, beyond a bachelor's degree. Including both academic and industrial professions in physics provided a broader perspective on the role of computation. Our subjects needed to meet at least one of the following criteria:

- Is an active or past researcher on computational education in a physics classroom.
- Is an active or past instructor of a computational physics course.
- Is an active or past instructor of an introductory physics course.
- Is a physics graduate who works or worked in industry.

Using these search criteria, we identified potential participants via web searches and created a list to invite them individually to be interviewed. A total of 26 participants accepted and were interviewed. Eight of these participants were physicists from industry while the remaining 18 were physicists in academia. Of the eight industry physicists, five have their doctorate in physics, one has their doctorate in astronomy and astrophysics, one has their doctorate in electrical engineering, and one has a bachelor's degree in physics. The industry physicist's occupations ranged from software engineer, experimental physicists in quantum computing, geoscientist, yield engineer, scientist in data informatics, and materials physics research. We did not collect any demographic information from our participants. While a diverse population is always preferred, we made a deliberate choice not to focus on the reporting of our participant demographic pool. This may have provided an additional insight into how CT is defined for different populations within physics, however, our goal is to determine the learning goals so that we may develop an assessment. It also would have increased credibility to provide information on the years of experience of the participants, but since we decided not to collect demographic information, this information was not obtained.

B. Interviews

The interviews were semistructured, about an hour in duration, and recorded and conducted online via Zoom. The interview questions focused on computation in the introductory physics classroom. For example, we asked, "What is the most important skill students can learn from a computational intro mechanics class?" and "What evidence would you look for to see if students met the learning goals in a computationally integrated intro mechanics class?" The full list of interview questions is provided in the Supplemental Material [39]. Of the 35 total interview questions, there were 24 that were asked to every participant, while there were a few that we only asked the industry or academic participants. For example, we asked industry physicists if programming is a skill they are expected to know, and we asked academic physicists about the classes they have taught before and how many times they have taught introductory mechanics. In total, industry physicists

were asked 31 questions and academic physicists were asked 28 questions. Most questions we thought of ourselves, but there were three questions that stemmed from a theory of computational physics literacy described by Odden *et al.* [40].

Before starting the interview, we defined the difference between programming and coding (you must write code to write a program, but you need not write a program to write code) and then stated "When we say computation, it will typically be in the context of computation in physics." We chose to not explicitly define computation and computational thinking to the interviewees for a few reasons. First, computational thinking is hard to define and we did not want to bog down interview time with us trying to define it for them. Next, providing a concrete definition would force our narrative onto the interview participants and bias responses. Finally, a majority of the interviewees work closely or have worked closely with computation in physics, and so they responded based on their own interpretation of computational thinking. Leaving the definition open ended, allowed the interviewees to use their own interpretations and definitions in their responses. Then, we were able to refine our own definition and understanding based on the interviews.

C. Analysis

The interviews were first transcribed and then imported into the qualitative analysis software program Nvivo 12 Plus [41]. Because our analysis methods involved *coding* participant responses about program environment *coding*, we distinguish the two interpretations of the word "code" throughout the rest of the manuscript as follows: code/codes/coding^q will represent qualitative codes used in the analysis of participant responses while code/codes/coding^c will represent the computational or programming side referenced by the interviewees.

In this paper, we focus on our analysis of the five question topics that are most informative for the development of an assessment: Important computational topics, class programming environment, limitations or difficulties, reasons for computation in introductory physics, and how to assess. We drew on methods of constant comparison and grounded theory to generate codes^q that capture common themes related to these topics [42,43]. Glaser describe four stages of the constant comparison methodology: Comparing incidents, integrating categories, delimiting the theory, and writing the theory [43]. They also describe that throughout the four stages of the constant comparative method, the researcher continually sorts through the data collection, analyzes and codes^q the information, and reinforces theory generation through the process of theoretical sampling. The benefit of using this method is that the research begins with raw data; through constant comparisons, a substantive theory will emerge [43]. Thus, this methodology not only shaped our interpretations of the data but also influenced our data collection process. The research team went back and forth between collecting and analyzing qualitative data, and comparing individual interviews to one another as they were being conducted to modify the interviews to focus on certain questions more closely tied to our research goals. For example, initially, we asked questions about the programming environment "scratch," but these were found to be not as useful and were dropped from the later interviews.

Once all interviews were completed, the first author devised an initial coding^q scheme and coded^q all of the interviews. Then, the first and second authors met to discuss any disagreements in the way the interviews were coded^q and refine code^q definitions. These authors also discussed common themes that emerged from the interviews and iteratively grouped the emergent codes under each theme.

Once a final coding scheme was defined, we calculated interrater reliability with a third researcher by having the first author, the third researcher go through 50 responses. Using a subset of 20 responses, we reached an agreement rate of 75%. Then, the first author coded the rest of the data.

IV. RESULTS

This section is organized around each of the five questions of interest (see Fig. 1), with an additional section about simulations at the end. We begin each subsection with a description of the interview question itself and then describe the themes and codes^q identified from the interviews, with examples.

A. Important computational topics

This question elicits computation-specific topics or ideas in introductory physics that either the research team or the respondents themselves deem important. We identified 44 different response codes^q for this question. These response codes^q fell under five broader themes: *technical programming skills and knowledge*, *programming environment best practices*, *physics and programming*, *barrier reduction strategies*, and *constraints*. Response codes^q could fit within multiple themes, but we place the response code^q with the theme we believe it most aligns with even though this categorization is not necessarily exclusive.

1. Technical programming skills and knowledge

This theme encompasses technical knowledge and skills believed to be used more often in the programming and computational science context than the physics context. The following individual codes related to this theme:

1. *Read Code^c*. This response code^q had the highest response rate out of all the response codes^q in this study. Of the 26 participants, 20 (77%) mentioned that students should know how to read code^c by the end of a computationally integrated introductory physics class. There was one participant who explicitly stated that students should not be expected to

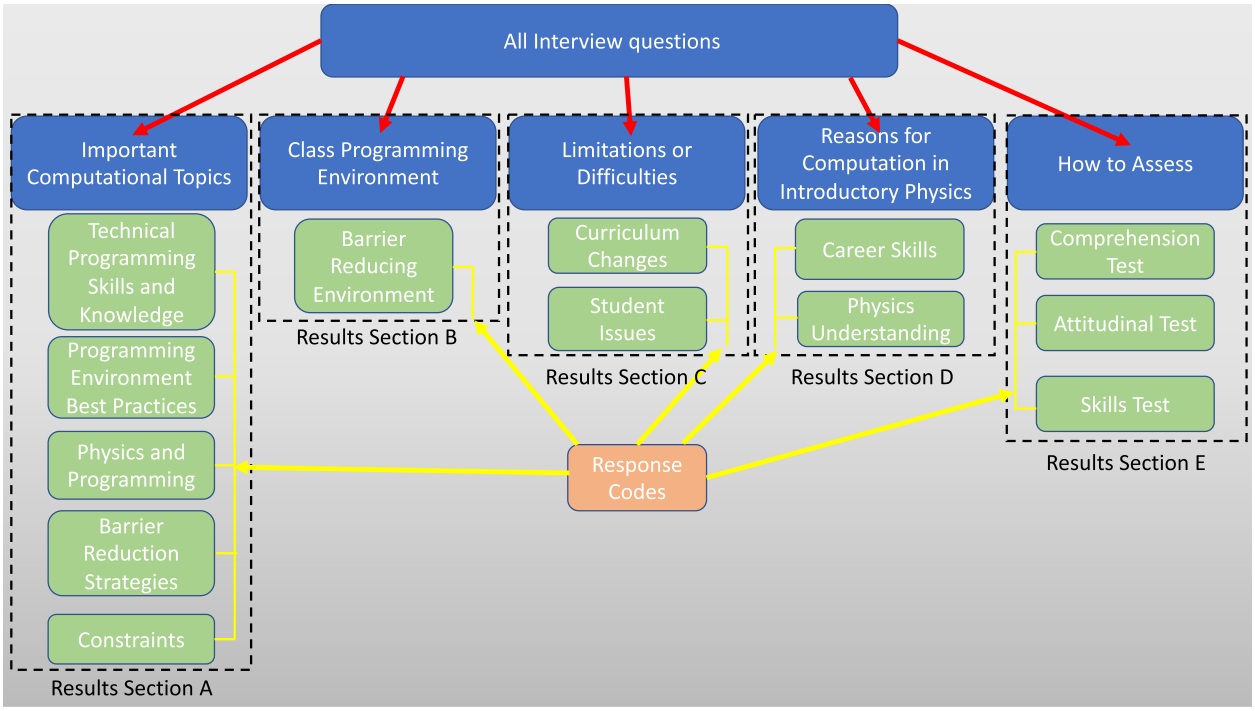

FIG. 1. General outline of our results, in the order we present them. Of all the 35 interview questions, 5 are highlighted in this paper (represented in blue boxes). For each of the five questions, themes were extracted based on the participants' responses (represented in green boxes). The orange box represents individual response codes[q] that fit into each theme. Responses could fall under multiple themes within a specific question.

know how to read code[c] by the end of the term. This participant wanted students to focus on the underlying numerical calculations performed. They wanted students to use the medium most comfortable to them and so that meant that not all students should be expected to know how to read code[c]. Almost all participants who mentioned that students should be able to read code[c], also added that students should only be able to read code[c] that they have seen before. Note that all participants were given pseudonyms to protect their identity. Ore provides a typical response from the interviews.

Ore: I would expect them to be able to look at a code and be able to read out the different parts. So you should be able to identify where in the code are objects defined, where in the code is your initial conditions that run the code, where you are doing your calculations, What kind of calculations this list code is doing, and then also be able to make some predictions about what they think the code should produce.

In Indiana's response, we can see how they talk about how it is unfair to ask students about "for loops" if they have spent the entire term learning and practicing with "while loops."

Indiana: Yeah, they should be able to read the code that was introduced to them in the course. So for example, I brought up the idea that some people won't introduce for loops. So if you hand them a for loop after they spent the whole semester only learning about while loops. That's not fair.

This is an interesting point in the development of the assessment. We do not want to assess students with programming code[c] that they have not seen before, but every class is going to be taught a little differently. There will need to be adjustments to the assessment with time to calibrate a base set of programming fundamentals that are universal to every class.

2. *Data Visualization*. Nine participants (35%) mentioned that plotting is an important skill to learn in an introductory computationally integrated physics course. Data visualization and data communication are good skills for all scientists as well as many other professions to know and learn. Data visualization also provides another representation of physical phenomena which can be helpful for student learning. Kerry points out that making and describing plots is an important skill that is not necessarily constrained to computational methods, while Lee describes how making plots and plot interpretation deserve a lot of attention in the course.

Lee: But yeah, to meet the minimum (learning goals) is lots of practice doing visual representation using the computer. So a lot of plots and a lot of interpretation.

Kerry: One could argue to what extent this is specific to computation, but once you have a program that's producing results: actually making plots and understanding plots and being able to describe plots. That's a very important skill, although that's a skill that also applies equally well to experimental measurements.

While data visualization and interpretation have been a part of physics, it seems that giving students more practice with the creation and interpretation of plots within a computational environment is a path to be explored more.

3. *For Loops and While loops*. Eight participants (31%) explicitly mentioned for loops or while loops as important skills to learn in an introductory computationally integrated physics course. Looping statements are typically the backbone of visualization of physical phenomena in motion or throughout time. A similar idea of *iteration* was mentioned by six participants (23%) as important, however, we kept these ideas as separate codes[q] because for loops and while loops are used specifically in programming environments, while iteration is a general concept used in CT. Another reason we kept these two separate is because iteration is a concept that can be explored in spreadsheets, while for loops and while cannot. Mason talks about how it is important to know which parts of a program need to be inside of a loop or not.

Mason: Since almost all these calculations involved iteration, can they understand how a loop works. Can they make us a small prediction about the functioning of the loop, can they explain why or why not you could move some lines of code out of the loop or not.

Something that was not nearly mentioned as much as the idea of computing time. There are plenty of variables that can included in a looping statement that do not change. It does not make the program wrong or produce an error, but it does increase the computational time required to finish the program. We postulate that most introductory physics are not introducing programs that are computationally intensive enough to affect the computing time in a noticeable way.

4. *Writing Code*[c] Seven participants (27%) mentioned that writing code[c] is important but did not always specify conditions (i.e., writing from scratch or writing from minimally working programs). Kerry describes writing programming as being important by relating it to playing the piano.

Kerry: I think with anything, you need to do some writing in order to really understand it; to really understand what's happening there. One could make the same argument with playing a piano, right, that you might be able to read the music. That's a useful skill, but that doesn't mean you'd be able to play the piano. So if you want to play the piano, you need to actually play some notes right. We're not trying to make great pianists out of the intro mechanics students, but they should be able to do more than just read the music. They should be able to play a few notes.

Kerry essentially says that students should be able to write some programming code[c] and that it is not enough to just be able to read it. They mention that reading is an important skill, but it just is not quite enough for these introductory students.

These four response codes[q] had the highest response rate: *Read Code*[c] (77%), *Data Visualization* (35%), *For Loops and While Loops* (31%), and *Writing Code*[c] (27%). There are a few other response codes[q] that we found interesting but not interesting enough to have their own subsection. Six participants (23%) mentioned *Euler or Euler-Cromer methods* as a skill for students to learn. The Euler method is an approximation method typically used in introductory computational physics because it is easy to use and provides a good enough result when modeling physical phenomena. Four participants mentioned *execution order* and *functions*. Execution order comprised responses involving how computers interpret commands in a certain order (first this, then that, etc). Functions was interesting because four participants advocated for the inclusion of functions as a skill to learn, however, it is almost entirely counteracted by three participants advocating that students should not learn functions (in constraints section). When designing the assessment, we will need to be sure to limit the number of functions used. This sort of makes sense because while some courses may use the same programming environment, they may not necessarily use the same functions. Other response codes[q] that are within this theme but did not have as many responses were *Conditional Statements and Arrays*, *Debugging*, *Spreadsheet Manipulation*, *Numerical Integration*, and *Step Size*. Numerical integration was not mentioned as often. This might be because as Chabay and Sherwood say, "Computers are now fast enough that it is not necessary to teach sophisticated numerical analysis techniques; simply using a very small time step provides adequate computational accuracy" [44].

2. Programming environment best practices

This theme encompasses the best practices used in programming environments. These involve cleanliness, communication, and structure within and of programs.

1. *Code[c] commenting*. Nine participants (35%) mentioned that students should comment code[c] and should know how to comment code[c]. This is a multifaceted and deeply rooted goal because it covers reading and understanding the code[c] and effectively communicating the physics within a program. There was one participant who stated that students should not comment their code[c], however, this participant then followed up to say that the code[c] should explain itself with meaningful variable names and not need comments. Aiden discusses how program documentation is important and that communication is a big part of this process as well.

   Aiden: And I think both of those are important. I think that documenting the code is important for any kind of programming. And then I would hope that this would be ideally built into the structure of the class where it would involve communication and collaboration as part of that.

2. *Meaningful variable names*. Eight participants (31%) mentioned that students should either know how to create and assign variables or that students should make meaningful variable names. Hunter describes well-written code[c] as being able to speak for itself. They talk about how commenting is still a helpful practice, but in the end, the code[c] and variable naming should be all that is needed to understand what is going on.

   Hunter: But definitely if you're working part of a large team, and if you're developing software that's going to be used by hundreds of people in perpetuity for many years, then I think it's critical. The last thing I would say is that I think commenting is not always the mechanism for making sure that code is understandable, and well written code can speak for itself, especially in a language like PYTHON, which is a very nice naturalistic syntax. Surely commenting often does help and enhance the codes ability to stand on its own.

3. *Intelligible code[c]*. Two participants specifically from industry mentioned that student code[c] should be intelligible: it should be clean, neat, and easy to follow. One participant stated,

   …it's basically that someone could come in after you and understand your code… I have PhD's where we still can't figure out what they did in their code, and they don't seem to care. Because they understand it, they don't care that their teammates can use it. So if you learn from the very beginning to have proper code hygiene and comment, everything should be logical. Someone else should be able to follow this because it's in a logical sequence. I think it becomes natural for you and you'll always be you're always write code that at least someone has a chance of following without having you sitting there next to them.

This participant notes that proper code[c] hygiene involves a logical sequence. This participant from the industry describes how sometimes Ph.D. students do not follow code[c] hygiene and how it can be detrimental to the team if the program cannot be passed on to someone else without the program's translator.

3. Physics and programming

This theme heavily focuses on physics and how physics intermingles with computational programming environments.

1. *Identifying core physics*. Nine participants (35%) found it important for students to be able to identify the core physics statements within a computational program. We show two participant excerpts from Zion and Lee, respectively:

   They should be able identify where it (the physics) is. So again it depends how you designed your course. If it is designed with 'I want to teach them physics and this is a tool,' then they should know how to use the tool… I know how to use a car, but I don't know how every bit of a car works. There is a gas pedal and a steering wheel. I know those really well…

Here Zion describes the use of programming environments as a tool for understanding physics. Zion relates this relationship of tool understanding to the operation and use of cars. Zion says that they do not understand all the inner workings of a car, but essentially they don't need to because those inner workings are unimportant. The real importance is in getting to the location Zion wants to be. This analogy in this context, we understood Zion as meaning that as long as students know and understand the physics in the program, the rest does not matter as much.

Providing minimally working codes or maybe minimally not quite working codes. It eliminates all of that barrier to just even getting started and then it's just concentrate on these lines right here, the physics is all in here. Do the physics. And yes, you do have to encode it, but take a look at the equations we've written down. And there's the translation. That translation right there is not easy I think for students. So for that to happen to concentrate on something that's not easy. That's what you want

them to do. You don't want them to say, 'Oh, I gotta comment this' and 'Oh, I missed a colon' or something like that. No. Put the cognitive burden, where it belongs.

Here we see Lee talking about minimally working programs. Lee goes on to say that students' cognitive burden should not be on the technical underpinnings of program syntax but instead on the physics. We believe that Lee is advocating the importance of focusing on the physics within a program out of everything else to make it easier for students.

2. *Modify physical systems*. Eight participants (31%) want students to at the very least be able to modify physical systems computationally after taking this course. This response code[q] ranges from things like changing numeric values in a program, as Kerry states, to extending existing programs as Mason states.

   Kerry: I think it is important and useful. And I think there's not much lost and a fair amount gained by having them see code and interact with code and change lines of code and then run a program. They're not going to become good programmers. That would be an unrealistic goal to say they're going to become good programmers as a result… So that's where they're taking some code that exists and they're running it and they're analyzing the results and they're understanding something from it and they're changing parameters. Yeah, that's very appropriate.

   Mason: …And really understand enough of the model that they can extend it. So, for example, asking them to calculate and plot kinetic and potential energy means they have to understand what the constructs in the code mean, how they would calculate kinetic energy or potential energy of the system, and then how can they plot it. Now for that part I let them look at their old programs so they can pull out graphing code or energy calculated code or whatever they want.

3. *Physics-program translation*. Seven participants (27%) advocated that students should be able to translate program notation physics into some other form or representation physics. Aiden describes how program notation physics is just another physics representation to learn. They talk about how it adds to their other representations like equations, graphs, and physical phenomena description. This is a place where reading comprehension may come into play. Is the reading and analyzing of a program considered reading comprehension or is it something else? This is similar to reading comprehension of a graph. Are graphs read or are their data interpreted? For now, we will consider the analysis and interpretation of programs as reading comprehension.

   Aiden: In the same way that in any intro mechanics class like we would want students to be able to translate between the description of a situation and equations and graphs and translating among these different representations. I think that computation would add one more kind of representation. So I think a goal would be for students to be able to take a physical situation and figure out how to translate that into an algorithm or into a program that they could run and vice versa, to be able to look at a program and figure out how that translates into to the physical situation.

3. *Justification*. Five participants (19%) advocated that students should be able to justify their program's output. This concept is not specific to CT as justification is an ever present skill to be learned in physics and STEM. In Riley's quote, we can see how it is more important to them that the students can explain and justify the results and output of the program. This notion is also seen in experimental physics labs.

   Riley: I want them to explain what the program does. I don't really care about the colon and all that stuff. Just tell me what the program does, and tell me how you know it's legitimate. That's why when they turn it in, I have them make a video where they show me their program and they just, show me how it works and show the output and show that's legitimate.

Some other response codes[q] in this theme were *output prediction*, *authentic programming practice*, and *system assumptions*. Respondents mostly focused on students being able to read code[c] and find the physics within the program. Along with finding the physics, participants wanted students to translate between program notation physics and some other representation. They also expect students to be able to modify a program in some manner and be able to justify or explain the output of programs.

4. Barrier reduction strategies

This theme represents practices or skills added to the course that are more focused on making CT easier for students to begin working with.

(1) *Scaffolding incomplete* code[c] Twelve participants (46%) mentioned including code[c] writing as a skill for students but only prefaced by the fact that scaffolding is in place.

> Gale: I firmly believe that the way you learn to write code is first writing, and filling in incomplete code. Filling in the blanks for incomplete code is scaffolding to be able to write the whole function or sentence or whatever yourself. It is much less intimidating than starting from a blank screen.
>
> Lee: Providing minimally working codes or maybe minimally not quite working codes. It eliminates all of that barrier to just even getting started and then it's just concentrate on these lines right here, the physics is all in here. Do the physics. And yes, you do have to encode it, but take a look at the equations we've written down. And there's the translation. That translation right there is not easy I think for students. So for that to happen to concentrate on something that's not easy. That's what you want them to do. You don't want them to say, 'Oh, I gotta comment this' and 'Oh, I missed a colon' or something like that. No. Put the cognitive burden, where it belongs.

With scaffolding, we can see that participants really care about students not being overwhelmed. It serves at least two purposes: (1) to make programming less scary for students, and (2) to focus student's attention on the part of the program that is important which is the physics. What we take from this is that any program writing a student is doing in this course should be set up in such a way that students are only adding or editing the physics concepts. This way the focus is on learning physics and not program syntax.

(2) *Learn real applications*. Six participants (23%) advocated for students to learn about, but not actually practice, real computational physics applications. The reason why they do not want students to practice these applications is because they are too complicated for an introductory class. Dakota talks about the value of learning some real applications.

> Dakota: I think if you're going to put that in computational physics or computational component to the introductory physics courses, it might be worth spending a lecture talking about the range of types of computations, even if they're not going to use the full range. There are analytical computations, but there are other things. There are Monte Carlo's. Okay, so they are not going to do a Monte Carlo calculation or programming up but they should know what a Monte Carlo calculation is and when you apply it. Nowadays neural nets are incredibly popular, especially in the industry. So they ought to know what a neural net is and how it works and why you would apply it, so there there are a number of computational tools in the toolkit.

Some other codes[q] that fit this theme were Pseudocode[c], *Experience Error Messages*, and *Exposure Comfort*. Pseudocode[c] is the idea of writing a program outline outside of the program environment. Typically, this is done with pencil and paper. When participants mentioned pseudocode[c], they described it as a way that presented CT and programming as a lower risk, lower fear way to get students thinking computationally. Experience Error Messages had a couple responses where participants wanted students to get used to seeing error messages that arise from programming. The hope was that students could build emotional resilience to the common frustrations involved with errors encountered while programming. Exposure Comfort had responses about students getting comfortable with programming environments by just being exposed to it. They talked about how using a programming environment as a calculator was a low risk way of getting them used to the way the environment and tool works.

5. Constraints

Many of the constraint theme response codes[q] have already been mentioned. This theme focuses on ideas or skills that participants do not expect students to know or are limited in some way. Constraint often implies a negative tone but know that is not our intention in this case.

1. *No code[c] from scratch*. Nine participants (35%) advocated that students should not be coding[c] from scratch in an computationally integrated introductory physics course. Participants who mentioned this also often mentioned scaffolding as important.

> Vesper: It makes a huge difference for student frustration level. So, especially at the intro level. I always give them some kind of structure where they have a couple lines, they have to fill in, but they're given that template and there are students in the class who would be capable of coding from scratch, especially if you build up to it, but that requires sort of a deeper understanding of code logic and having a deeper level of mastery of coding and I don't have enough time in class to give all my students those things. and some of them are chem majors who are not taking computer science. So I don't even attempt to have them write code from scratch in intro physics ever.

Vesper says that they never have their students write from scratch ever because it involves a deeper level of understanding that they do not have time to delve into. They also mention that it helps with keeping the students from becoming frustrated.

> Xoan: so we give them Templates. It's not quite the right word. Incomplete programs to start from scaffolding. So some of the basic stuff is there already. But we insist on having them do the key physics statements that are in those programs, which in the case of a PYTHON program is a significant fraction of the number of lines unlike many programming environments. But it's still they cannot write from scratch, and we were sort of given up that as a goal.

Xoan mentions that they have given up on students writing code[c] from scratch as a goal insinuating that at some point, they tried incorporating that as a goal and later decided against it.

Earlier in our important computational topics sections, we showed that some participants mentioned writing code[c] as important, but they did not mention whether it should be from scratch or not. Assuming all the participants earlier were talking about writing from scratch (seven participants), these nine participants saying students should not write from scratch entirely counter them.

2. *Programming is supplemental*. Three participants mentioned that programming is not the focus of the course and that it is purely supplemental to learning physics. Mason bolsters the idea that physics is the main topic to learn and that computational methods are just a means to that end.

> Mason: Any computation things they're doing should be in the service of deeper learning and understanding of physics.

3. *No functions*. Three participants said that functions should not be learned in intro physics. This essentially counters the four participants who said that functions should be included.

B. Class programming environment

This question had 109 responses, with all 26 participants contributing to it. This question represents the programming environment(s) or language(s) that participants believe should be used in the introductory physics course. This question is not broken up by theme because all responses fall under the same theme: The environment that has the lowest barrier for students. A programming environment is the application that the programming is conducted within, while the programming language is the specific way of writing instructions for the computer to run.

1. PYTHON and VPYTHON

PYTHON (20 participants, 77%) and VPYTHON (14 participants, 54%) had the highest response rate in this category. PYTHON and VPYTHON are free, open-source programming languages. VPYTHON is short for Visual Python and has a focus on animation and visualization of code[c]. While these two had the highest response rate, they are languages and not environments. Some environments that were mentioned that use these languages but did not have as high of a response rate were *Glowscript* (six participants), *Jupyter Notebooks* (two participants), *Trinket* (four participants), and *Spyder* (one participant). Glowscript and Trinket are both web-based with no local file storage and use VPYTHON. Three participants describe why they advocate for PYTHON/VPYTHON:

> I exclusively use PYTHON and VPYTHON. Just because it has become the most accessible language and most popular language. I guess all over the world. And it's easy to get help. It's easy to get set up with PYTHON more than with other languages. So I guess I'd vote for PYTHON and the PYTHON.
>
> And most physics labs are equipped with PCs or Macs or tablets or laptops. So even if students don't have devices that they can do the computation on, all these are codes that run in a few seconds. So, you know, there's no reason why they should not be able to do them. And if you're using a browser based platform like Glowscript or you know Trinket there is no added problem or challenge of installing software. So the hurdle is less or minimal.
>
> So for intro mechanics, the most likely one that I would expect to see would be using Glowscript, VPYTHON on because from a logistical point of view, it's easy, that it's just you running into web browser without having to download things and you get visualization built right in.

We see participants describing how these languages and platforms are more accessible to students as they do not need to deal with downloading software. This means that they can run computational programs on any device as long as they have Internet access. This is a great option for students who might share a computer or who do not have a computer they can reliably use. There were two responses that advocated for not using VPYTHON or Jupyter Notebooks. The main argument for them was that VPYTHON would not be used later on in their career and that Jupyter Notebooks were too much to learn for introductory physics students.

2. Spreadsheets

Spreadsheets were the next highest response rate (11 participants, 42%). The most common spreadsheet

environments mentioned were Microsoft Excel and Google Sheets. Participants described spreadsheets as serving as a noninvasive way to teach CT and that most students have already used or heard of spreadsheets so there is less apprehension toward learning or doing physics in them. Another reason participants advocated for spreadsheets was that students could actively see iterations and how data change with each iteration. They also note that students will likely be working with spreadsheets in the future so more experience does not hurt. Participants Frankie and Paris talk about how they use spreadsheets for CT.

> …I do sometimes use Excel or Google Sheets, just because when you code and you use loops in PYTHON or any other language, what happens in the loop is pretty much a black box. So you don't see the intermediate numbers being generated step by step. So it's useful to demonstrate that through a spreadsheet. So, for the Euler method for every delta t increment, you can see how the velocity changes or how the acceleration changes or how the position changes and whatnot…
> I'm also partial to Excel programming. Because a lot of students may not have come in with previous exposure to PYTHON or other programming languages. But it's far, far more likely that they've come in with exposure to Excel or other spreadsheets, not necessarily programming in it. They may have used it just to like organize their books or keep track of like business finance it your family finances or whatever. But at least they're familiar with that software so that when you start introducing ideas of iteration into it. There's a little bit less of an overhead to that.

3. Language exposure nonspecific

The third highest response rate was that the *language exposure be nonspecific* (10 participants, 38%). Essentially, participants stated that as long as students were getting practice with a programming environment, it did not matter which one they used. Dakota says that programming environments change over the years so focusing on learning a language is not for the language itself, but learning the ideas associated with using a programming environment: Computational thinking.

> Dakota: Yeah, that's a tough one, if there's one language that's been used for the course then the answer probably is yes, but the reality is, languages come and go… I think I tell people the language is going to change the language that they're going to use five years from now isn't going to be any of those. There's no reason to get too caught up in the specifics of the language.
>
> Elliot: As long as you can program I think picking up another language is not such a huge problem. I think it's maybe learning your first programming language, and making sure that you have one.

Some other similar response codes[q] to this one were *Easy or Quick to Learn* (five participants) and *Knowledge of Multiple Languages* (two participants). The Easy or Quick to Learn response code[q] was again language nonspecific with the stipulation that the language used be the easiest language to learn. The Knowledge of Multiple Languages response code[q] described the idea that students should learn more than one language in introductory physics. Their reasoning is that the more languages students learn, the more computational thinking strategies students can learn and can be applied to any language.

4. Other languages

There were other languages that were brought up, however, none of them were brought up as frequently as PYTHON or VPYTHON. The next highest language was MATLAB (five participants) but every other language had three or fewer participants. There were, however, three participants who mentioned that students should not learn C/C++ and three that mentioned that students should not learn JAVA. Both of these environments were described as being overly complicated compared to PYTHON/VPYTHON.

C. Limitations and difficulties

This question represents the struggles of adding computation into an introductory physics course. Knowing what to expect in terms of roadblocks to integrating computation into introductory physics is important so that aspiring instructors know what things to look out for. This question code[q] had 21 participants contributing responses. These limitations and difficulties range from curriculum changes that need to be made to student attitudes toward computation.

1. Curriculum changes

This theme has the two most cited difficulties: *No Room for Computation* (14 participants, 54%) and *Curricular Overhaul* (8 participants, 31%). The most cited difficulty is that the introductory physics class does not have the space to learn computation as well as physics. Of those 14 participants, 3 of them were from industry so it shows that industry physicists are even thinking of this. How can computation be added to an already full content physics course? This difficulty, while a valid concern, was also addressed by most of the participants who mentioned it. They mentioned the idea that some things would have to be removed from the course in order to add computation, however, the things removed are gained back in a better fashion through the implementation of computation. Jesse

talks about this: "…Another limitation that gets brought up is that integrating computation will come at the expense of other material, sort of time on task spent solving analytical problems or covering additional concepts or what have you. I feel like that's bunk because if you integrate computation in the right way, you don't get so much of that loss, and in fact, it can simplify certain aspects of the physics teaching and learning like planetary motion. For example, it's a lot easier to simulate planetary motion and explore Kepler's laws through a computational simulation than it is to go through the full complete analytical derivation and all of it's intense glory".

The second difficulty is the response code[q] *Curricular Overhaul*. This is the idea that in order for computation to be effectively introduced into the introductory physics course, an entire overhaul of the curriculum is needed to integrate most of the content with computation.

> Aiden: It's to do computation. Right. It's not something that can be done with just throwing in this one assignment here. It's something that in the classes that do it well, they really build it up over the course of the whole semester. So it feels to me like more of a sort of all or nothing that if I'm going to do it, I need to really overhaul the whole class to do it, which is not something that I've done yet. I mean, if there are ways to do it in smaller doses then I would do that.

Just like how adding simulations to the course does not make the course a computational course (see Sec. IV F), adding some computation without overhauling the entire curriculum is not an effective way of making the course computational. This then requires that to transition from introductory physics to computational introductory physics, instructors will need to spend a good portion of time overhauling their curriculum to include computation in most aspects of the course. This is a very time-consuming task that is not always incentivized.

2. Student issues

This theme focuses on issues that are more specific to the students than the class or curriculum itself. The three highest response codes[q] in this theme are *Student Rejection* (seven participants, 30%), *Differing Levels of Computational Literacy for Incoming Students* (six participants, 23%), and *Technology Accessibility* (four participants).

Student Rejection. Talks about students will not accept computation in the course. Here are some excerpts talking about this:

> There are people though, who I think there are students who would resist it, because they would think to themselves, well I sign up for a physics course, not a computational course. So you definitely have to be careful about the context and the support.
>
> But there's also a lot of maybe fear or anxiety around programming. That when students are given code if they haven't seen it before, then that can be a barrier for them.
>
> The only difficulty there, especially with the pre meds, is that the pre meds have to pass this thing called an MCAT exam. And they are expecting a physics problem solving bootcamp when they wander into our classes and anything innovative like using a spreadsheet or programming language to solve problem isn't going to help them because they're purely there to pass an MCAT exam.

In the first quote, we see the participant talking about how students would resist the course because the students view computational practices and physics as separate and nonoverlapping things. The second excerpt talks about how students have a fear or anxiety toward programming. This fear is described as a barrier for students and that barrier can lead to students shutting down and rejecting CT practices. The third excerpt is interesting because it discusses the MCAT. The Medical College Admission Test (MCAT) is a standardized, multiple-choice examination that is a prerequisite to the study of medicine [45]. They discuss how these students who are only taking a physics class to prepare for the MCAT are not interested in computational methods. Does the integration of computational methods and CT into an introductory physics class have any effect on the physics portion of the MCAT for these students? Research will need to be done to shed light on this area.

Along with student rejection comes the idea of *Differing Levels of Computational Literacy* for incoming students. Every student has a different knowledge level when entering this course. Because computational practices are becoming more prevalent even at the high school level, some students have worked with computational environments before they reach introductory physics while others have not. Just like how students have differing levels of physics knowledge coming into the class, practices will need to be put into place to make sure that any student regardless of background and prior knowledge can excel in this course. Jesse provides their take on this: "There's a limitation in terms of preparation. I feel like this starts to touch on a bit of an equity angle as well. If you don't spend so much time teaching the students computation, then those who come in with computation already in their back pocket, who are typically the students coming from these, these well resourced high schools or, you know, families with parents who are engineers or programmers, that kind of thing, those students will have an inherent advantage. That I do think is a legitimate complaint and a legitimate limitation here. And something that the instructor of course themselves is going to have a hard time addressing unless you know you offer students a whole bunch of extra sessions to brush up on their computational skills or the computational

literacy, which most instructors don't have the time for. So really, that's something that has to kind of be addressed before the students even arrive at the university like what Norway is doing right now with their integration of computation across the high school curriculum".

Jesse views this student limitation from an equity lens. They also bring up what Norway is doing. Norway is integrating CT and programming into their mathematics, music, social sciences, and English classes in an attempt to emphasize CT and programming in a multitude of disciplines and incorporate CT and programming as a core subject itself [46]. Caballero *et al.*'s 2012 paper found no statistical difference between the performance of students who took the class with prior programming experience and students with no prior experience [37]. It could be that this is true of other courses, and so it may not be an issue. Also if other countries follow suit with what Norway is doing, then that may fix this issue as well.

The last response code[q] in this theme is *Technology Accessibility.* This is not only a student issue but also a course issue as well. To program, students will need a computational device or computer to do so. This means that either students need to have this device themselves or the course needs to provide them. While computers are quite prevalent now, not every course or student has access to them so it is important to provide them. This is probably not a common problem, but when it is a problem, it can be a major issue.

> Uri: Well, one limitation of course is both hardware and software. Students have to have hardware, they're going to work at home, they have to have hardware at home. Not all of them can afford it. Same with software.

Uri talks about how students will need to have the hardware and software at home if they are going to work on these computational problems. They talk about how we need to be cognizant of students' financial situations. Even if students can use a computer on campus, this may prove to be a burden for commuter students.

D. Reasons for computation in intro physics

This question represents the reasons why computation should be added to the intro physics class. We do not spend too much time on this because it has already been discussed why computation should be added to physics classes in the introduction section. That being said, we will briefly discuss some reasons why our participants found it important.

1. Career skills

About 16 participants (62%) mentioned the idea that having computation in intro physics is important for building *career skills*. Many of the participants believe that the skills learned in this class will be used in the students' future careers. There was also the idea of this being a *21st century scientist skill* (six participants, 23%). Of the 16 participants who contributed to this code[q], 6 of them were physicists from industry. As there were only eight participants from industry, this became a little interesting. There is definitely bias because these physicists either work with computational practices or were computationally adjacent, however, one even mentions that it is not necessarily a skill just for physics:

> Yes, I think that would be very beneficial. Just to help even students making the connection between how a computer works which can be very useful for jobs is even outside of physics. It's if they go work in technology in a bank or they go work in other areas. It just helps to have that connection early on.
> Well, yes. I think it should be somehow because it's an important part of doing real physics. Computation ultimately is going to be something that even if you don't do it yourself, you're going to need to know. And know enough about it that you can appreciate what's going on and you know, maybe even have a chance to collaborate with people who that's what they do… You know, whatever branch of physics to go into is going to be very complex and likely going to need some kind of computation to really do." This industry participant talks about how it is an important part of doing real physics. They go on to mention that even if students don't do computation themselves, they will likely be around it and so having the knowledge to be a part of those conversations is useful.

2. Physics understanding

The next few response codes[q] mostly involve student understanding of physics as the reason to include CT. They are *Understand Physics Better* (12 participants, 46%), *Modeling Physics* (11 participants, 42%), *Skills Useful for Later Physics Courses* (10 participants, 38%), *Allowance of Non-trivial Phenomena* (10 participants, 38%), and *Creativity* (7 participants, 27%). Most of these involve bolstering physics concepts and practices among students. It makes sense that the main reason for adding CT is to further increase students' physics understanding overall.

E. How to assess

This question was asked near the end of the interview as a way to solicit ideas for the development of our assessment. The participant responses were broken up into three different themes: Comprehension test, skills test, and attitudinal test.

1. Comprehension test

About 14 participants (54%) talked about how they would assess for learning of CT by having students read code[c], comment code[c], or find the physics inside code[c].

> Frankie: That is a part of the learning outcome that what I meant by them being able to interpret what the code does is, and I assess it by looking at the comments that they wrote.
>
> Mason: So our minimum expectations are that students should be able to read and interpret a very short program that instantiates a physical model.

This seemed to be the most agreed-upon method and is a measure of students' comprehension more than anything else. Another response code[q] in this theme is predict and explain results (nine participants, 35%). This entails students predicting the outcome of a program or explaining the results of a program.

> Noel: I think it would be important to learn the assumptions one makes in setting up a computation. It can make the difference whether the result is reasonable or doesn't make any sense at all. I think often it seems to be easy to come to the conclusion that because the computation gave you a result that it must be true without trying to figure out ways to test them. In a sort of like a design of experiments, try different parameters, and making adjustments and seeing if the expected change and results is consistent so that you can say something more about whether the results that you're getting is actually believable.

These responses track with what has been discussed so far. Many of the participants would assess via reading, commenting, finding physics, or justifying a program output. Most of these have a focus on physics but in the context of a computational environment. We have noticed an emphasis on physics being important and the programming being supplemental to learning physics. Assessing students in this way seems to meet all of their expectations of not having them too engrossed in the syntax of coding[c].

2. Skills test

About 4 participants (15%) said they would assess students by having them write and run a program. This assessment style is both comprehension and skill based. Here is an excerpt from

> Gale: So I think it's a performance assessment. You give them a task and they have to encode that and send you the code and hopefully when you hit run it will compile and run. But yeah, you might be able to assess aspects of coding you know, in a written assessment, but No, there's no substitute for the real thing.

Gale says there is no substitute, in terms of assessment, for the real thing which is writing and running a program. Many of these four participants also mentioned that the students do not need to write and run a program from scratch. This skills test is harder for students holistically as they are combining many different aspects of CT and physics. On top of that, there are many different places where a student can go wrong and make it so that their program does not run. It is not great for partial credit in that sense because if the student gets stuck debugging part of their code[c], they cannot exactly move on until it is fixed. Parsing through student code[c] to find their issues is also no easy task as well. It can be quite difficult debugging or grading multiple student computational assignments.

3. Attitudinal test

About 4 participants (15%) said they would assess students by checking to see if the students have a better outlook or feeling toward computational practices.

> Paris: So I guess what I would look for is just more of a classroom climate aspect. Like, does it seem to me that students are taken to this or excited about it or maybe the negativity of are our students actually rejecting it or they at least kind of presenting themselves saying like 'yeah I guess this is what science does. So we have to do that.' Obviously I prefer them to be excited about it, but considering the spectrum of people who go through these courses, I think that's maybe one thing to look for is just is there active rejection or not. That's the hard thing to really measure and assess especially when most of classroom assessments are geared towards you know content assessments.

Paris talks about how they care most about how students feel about CT and physics together. They talk about how they hope for excitement from the students. This is important because excitement is a form of interest and that is related to persistence in physics [47]. They talk about how most assessments are geared toward content and view attitudinal assessments as harder to administer because of this. If we want physics to be an inclusive place, student attitudes become important to analyze.

F. Simulations

One of the things we sought out was to learn more about whether simulations were considered computation or not. We found that while simulations are computational in nature (made from computational practices), they indeed were not considered one of the computational skills students should learn. One excerpt that we feel highlights what most respondents said was this one:

> …especially with that simulation, you are very far away from what's happening under the hood. You don't see how the computer is actually working. For a [physics] major I think it's important to see that there's a few important lines of code here that implement basic physics that we want you to know about and you should be able to go in and make changes to a couple of lines of code in order to generate different results.

Respondents often said that simulations are great for learning. It is an excellent learning tool that any student can use to supplement their understanding of physics [48]. The main point that was often brought up was that simulations were not enough for physics majors. For physics majors, respondents wanted them to see, create, and manipulate the physics that would make a simulation. In most simulations, students do not have that kind of access. This was a very important point for our participants. There has been a push to include computation in physics classes, and so many curricula have added simulations to meet these criteria. What we have learned from these interviews is that simulations are not enough if the course is for physics majors. Simulations are not bad, they are great learning tools and should be added to physics courses, but they do not give students the opportunity to learn the computational skills that most participants mentioned to be important from these interviews. The Next Generation Science Standards view using simulations in science class as a clear example of computational thinking so it is interesting that it was seen as not enough by some of the interviewees [3].

V. DISCUSSION

The goal of the interviews was to discern what student skills or ways of thinking are important to learn after taking a computationally integrated introductory physics course so as to come closer to answering (RQ1). We attempted to answer this by interviewing 26 physicists and asking them about CT in introductory physics. We used constant comparative method, grounded theory, and emergent coding[q] on these interviews to pull out codes[q] and themes. We analyzed and presented five question topics deemed important from these interviews: Important computational topics, class programming environment, limitations or difficulties, reasons for computation in introductory physics, and How to assess. Within important computational topics, we found that Reading Code[c], Data Visualization, Code[c] Commenting, Identifying Core Physics, Scaffolding Incomplete Code[c], and Not Coding[c] from Scratch were some of the most prevalent response codes[q]. Vieira *et al.* examine the role of comments in computational education [49]. They found that students who did not understand the material well would use comments as a learning opportunity. Perhaps, this is why comments were viewed as important in our interviews. Within class programming environment, we saw that PYTHON and VPYTHON by far were the preferred languages to be used in introductory physics. The reasons for PYTHON and VPYTHON tend to follow the same reasons outlined in Backer's paper in 2007 [50]. We also saw that spreadsheets were also brought up frequently as an acceptable environment. Even though MATLAB was not mentioned as frequently, Flannery discusses an implementation of computational physics using MATLAB. One benefit of MATLAB that Flannery talks about is the one-to-one equation to MATLAB statement translations [51]. As translating physics into code and finding physics in code was viewed as important, MATLAB could be a suitable environment to hone this skill. Within limitations or difficulties, we saw that changing the curriculum was the most cited difficulty in integrating computational practices into the course. As Young *et al.* found, it is often the choice of the instructor to incorporate computation rather than an institutional or department-level decision [52]. This sort of isolation may contribute to the difficulties of adding computation to a course. We also had a few responses that focused on the student perspective and how they might have difficulties with accessibility, differing levels of incoming literacy, and rejection of material. Magana and Coutinho also found many similar themes in their interview analysis on modeling and simulation practices for CT-enabled engineers. They also found that their interviewees were worried about students' incoming math and programming skills [53]. Within reasons for computation in introductory physics, we saw that career skills were the most cited response with student physics understanding following after. Landau argues that physics discipline specific knowledge is not as important as the computational skills students can learn because there are more occupations that can make use of the skills than the discipline specific knowledge [54]. Chabay and Sherwood talk about how computer modeling can help students make concrete visualization of abstract quantities [55,56], and Macdonald *et al.* also believed that computers could help emphasize the fundamental physics [57]. While it is often not the subject of introductory mechanics, there are many aspects of physics that are either invisible, or not visible to the naked eye, and computer modeling can help students visualize phenomena to help with their understanding. Within how to assess, we saw participants discuss ways on how they might assess CT in intro physics. They mentioned attitudinal tests, skills tests, and comprehension tests. Comprehension tests had the most responses.

In Weller's framework, we notice a few similarities like translating physics into code, adding complexity to a model, analyzing data, and demonstrating constructive dispositions toward computation that were also found in our interviews [38]. Debugging was something that only three of our participants mentioned. This has its own category in Weller's paper so we were surprised to see that it was not mentioned more in our interviews. Other CT practices were not explicitly mentioned either. Highlighting and foregrounding were not mentioned by our participants explicitly, however, our participants did describe some processes of highlighting and foregrounding like making system assumptions, understanding execution order, and practicing pseudocode.

One overarching theme we notice from the interviews is the idea of students reading and understanding program code[c] as a learning goal. In the interviews, codes[q], and themes, we see that students Reading Code[c] is the most agreed upon skill to learn. Reading Code[c] permeates through many aspects of the response codes[q] from the interviews. Another overarching theme we notice is the general barrier-reducing strategies mentioned. Many participants mentioned that CT in physics can be daunting. They brought up ways and strategies instructors could attempt to mediate this via scaffolding incomplete code[c], focusing on physics rather than computational environment syntax, and using computational environments and languages that are more accessible. The learning goals are still the same introductory physics learning goals. Additional learning goals for CT in introductory physics are needed though, and we have seen that they mostly take the form of computational reading comprehension.

We would be remiss not to mention that part of the reason that many participants talked about students reading code is because we specifically asked them if students should be able to. It is important to note that it was explicitly brought up in the interviews by the interviewer. Most of the questions were open ended and not necessarily prompted about a specific facet of CT. For example we asked participants about programming environments in general and not PYTHON or "spreadsheets." All of their responses (to this question) are more valid in the qualitative sense because their responses were not steered by the interviewer. Even though participants were explicitly asked about reading code, we believe the results are still valid because aspects of reading code were present in participants' answers to other questions. We hypothesize that if we did not ask explicitly about reading code, it still would have come out of the interviews thematically.

Twelve participants mentioned something about scaffolding incomplete code[c], nine participants mentioned not coding[c] from scratch, eight participants mentioned that students should be able to modify physical systems, and seven participants mentioned that students should be able to write code[c]. What we can see from these interviews is that participants generally expect that students should be able to use a programming environment to at the very least edit a program with physical phenomena. Almost all of these participants were very adamant about how students should be writing their code[c]. Some reasons were that they were very conscious of how coding[c] and programming were perceived by students. They recognized that programming and coding[c] are viewed by students as scary or only for computer people, so to reduce this sentiment, they focus on having students only work on specific parts of the program rather than the program as a whole. This is often seen as better for students since they can essentially avoid the less fun and potentially deterring aspects of coding[c] like syntax and debugging and can in turn focus on the physics that they likely have more of a passion for.

A lot of these responses from these four codes[q] elude to minimally working programs. Minimally working programs in the physics context have been adapted slightly to fit their goals. Typically, a minimally working program would involve a short program with one small focused bug to be found and corrected by the user. In its adaptation to physics, it is now typically a full working program but with incorrect physics to be found and corrected by the user [58].

Another selling point that Mason makes is that coding[c] in real practice does not involve coding[c] from scratch. Often times, we will be building off of someone else's work or our own, and so starting from a blank program is rare. It makes sense that participants do not want students writing from scratch if it is a practice that they do not do themselves. Mason mentions that this is something that the pros do.

To answer our research question, we see that the learning goals do not differ much between an introductory physics class and a computationally integrated introductory physics class. The main goal of the course is for students to learn and practice physical principles. In the computationally integrated course, the way students practice is different since they are using a computational environment as a tool. While there are some base skills and practices that are more computational than physical (learning for loops and while loops, commenting code, creating variables), students are mostly expected to learn another representation of physics. This suggests that our assessment should focus on the base CT skills needed for students to modify physics in a minimally working program and focus on the intricacies of switching between various physics representations and the program physics representation. From our interviews, we see that this assessment should be written using the language of PYTHON, VPYTHON, or spreadsheets. The assessment should also be written and presented in a form that is less intimidating for students so they can continue to learn physics confidently.

VI. FUTURE WORK

Our future work involves a computational physics reading comprehension assessment. This assessment will be mostly multiple choice and multiple-choice response. In terms of determining how we can design an assessment for the computational learning goals found from these interviews, the most "concrete" learning goal we can test is reading code. This is also bolstered by most participants mentioning that they would attempt to assess CT in introductory physics via comprehension test. While skills tests and attitudinal tests are also feasible and need to be researched, based on this study, we have determined that we will design a comprehension test via reading program code.

VII. LIMITATIONS

One limitation of this work is the lack of demographic information on the participants. We describe our recruitment process, but ultimately it would have been beneficial to provide more information on the participants like years of experience teaching introductory physics, years in the workforce, and institutional information. One of the challenges we encountered was recruitment of industry physicists. We drew from the APS database of industry mentors. We agree it could be interesting to identify graduates of physics programs who are working as data scientists, but they are hard to find and not centralized. Another limitation is that we did not have a working definition of CT in physics when we conducted these interviews. We decided to develop our definition based on the interviews. The interviews could have gone very differently if we framed them in terms of a provided definition.

VIII. CONCLUSIONS

Another form of assessment that should be explored is the attitudinal test. While this may not be in the scope of this project, it will be important to design an attitudinal assessment for student sentiments toward CT in introductory physics. The reason this is important is because computational science, CT, programming, and physics are associated with the nerd-genius stereotype in STEM as described by Starr [59]. Adding CT to physics makes sense for bettering student understanding and learning practical skills for their future careers, but it does not help in the long run if it is also deterring and pushing out certain demographics. Shoaib *et al.* interviewed 8 men and 20 women engineering students about their identity with computation and engineering [60]. They reported incongruence between perceived feminine norms and computational identity. They state that students' computational identities can be supported through the intentional mitigation of bias by their instructors. Blaney and Stout examined computing self-efficacy and sense of belonging for first-generation women [61]. They showed that first-generation women have very negative computing self-efficacy and sense of belonging in computing compared to their counterparts (non-first-generation women, first-generation men, and non-first-generation men). They suggested that increasing instructor and student time both in and out of class can help increase sense of belonging promote high self-efficacy. It is important that we can analyze students' attitudes toward introductory computational physics so that we can make changes to create an inclusive and welcoming environment for any person who wants to learn physics. Computation is very much a real and important part of physics. As many have said before, in physics, it is the third pillar alongside theory and experimental. Computational practices will become more and more integrated into physics as time progresses. CT in general will become more integrated with STEM as a whole as time progresses. It is important that we can learn from other disciplines about how they integrate and define CT. Learning how CT is being integrated in the K-12 sphere is also important. With students getting earlier exposure to CT, it will become less of a burden for physics instructors to teach both physics and basic computational principles.

With these interviews complete, we will now work on a draft assessment of CT in introductory physics. The assessment, informed from this study, will be a computational reading comprehension assessment. We will focus on questions that show if students can read a physics program written in VPYTHON and can identify the physics principles involved, translate between algebraic notation and program notation, and explain if the results of a program are reasonable physically. Because we wish to focus on student comprehension more than a skills test, the assessment will not involve the creation, editing, manipulation, or extension of a program in a programming environment. Instead, the assessment will take on a multiple-choice style to concentrate on reading comprehension. This assessment will need to be validated and tested. When the assessment has reached acceptable validity, we will then pilot the assessment to student populations.

[1] J. M. Wing, Computational thinking—The beginning, Commun. ACM **49**, 33 (2006).

[2] K. R. Roos, An incremental approach to computational physics education, Comput. Sci. Eng. **8**, 44 (2006).

[3] NGSS Lead States. 2013 Next Generation Science Standards: For States, By States

[4] M. Guzdial, Software-realized scaffolding to facilitate programming for science learning, Interact. Learn. Environ. **4**, 1 (1994).

[5] S. Basu, G. Biswas, P. Sengupta *et al.*, Identifying middle school students' challenges in computational thinking-based science learning., Res. Pract. Technol. Enhanced Learn. **11**, 13 (2016).

[6] N. Hutchins, G. Biswas, L. Conlin, M. Emara, S. Grover, and S. Basu, Studying synergistic learning of physics and computational thinking in a learning by modeling environment, in *Proceedings of the 26th International Conference on Computers in Education* (Asia-Pacific Society for Computers in Education, Philippines), https://par.nsf.gov/biblio/10110253.

[7] S. Grover, G. Biswas, N. Hutchins, C. Snyder, and M. Emara, Examining synergistic learning of physics and computational thinking through collaborative problem solving in computational modeling, in *Proceedings of the Annual Meeting of the American Education Research Association*, https://par.nsf.gov/biblio/10110249.

[8] N. M. Hutchins, G. Biswas, M. Maroti *et al.*, C2STEM: A system for synergistic learning of physics and computational thinking. J Sci Educ Technol **29**, 83 (2020).

[9] J. Weber and T. Wilhelm, The benefit of computational modelling in physics teaching: A historical overview, Eur. J. Phys. **41**, 034003 (2020).

[10] AAPT Undergraduate Curriculum Task Force, *AAPT Recommendations for Computational Physics in the Undergraduate Physics Curriculum* (2016).

[11] R. F. Martin, Undergraduate computational physics education: Uneven history and promising future, J. Phys. Conf. Ser. **759**, 012005 (2016).

[12] M. Obsniuk, P. Irving, and M. Caballero, A case study: Novel group interactions through introductory computational physics, presented at PER Conf. 2015, College Park, MD, 10.1119/perc.2015.pr.055.

[13] P. W. Irving, M. J. Obsniuk, and M. D. Caballero, P3: A practice focused learning environment, Eur. J. Phys. **38**, 055701 (2017).

[14] N. Hawkins *et al.*, Examining thematic variation in a phenomenographical study on computational physics, presented at PER Conf. 2017 Phys. Educ. Res. Conf. 2017, Cincinnati, OH, 10.1119/perc.2017.pr.037.

[15] P. C. Hamerski, Daryl McPadden, Marcos D. Caballero, and Paul W. Irving, Students' perspectives on computational challenges in physics class., Phys. Rev. Phys. Educ. Res. **18**, 020109 (2022).

[16] M. D. Caballero, N. Chonacky, L. Engelhardt, R. C. Hilborn, M. L. del Puerto, and K. R. Roos, PICUP: A community of teachers integrating computation into undergraduate physics courses, Phys. Teach. **57**, 397 (2019).

[17] T. K. Holloman, W. C. Lee, J. S. London, C. D. Hawkins Ash, and B. A. Watford, The assessment cycle: Insights from a systematic literature review on broadening participation in engineering and computer science, J. Eng. Educ. **110**, 1027 (2021).

[18] C. Wyatt-Smith and L. Adie, The development of students' evaluative expertise: Enabling conditions for integrating criteria into pedagogic practice, J. Curric. Stud. **53**, 399 (2021).

[19] D. Weintrop, Elham Beheshti, Michael Horn, Kai Orton, Kemi Jona, Laura Trouille, and Uri Wilensky, Defining computational thinking for mathematics and science classrooms, J. Sci. Educ. Technol. **25**, 127 (2016).

[20] V. J. Shute, C. Sun, and J. Asbell-Clarke, Demystifying computational thinking, Educ. Res. Rev. **22**, 142 (2017).

[21] S. C. Kong, H. Abelson, and M. Lai, Introduction to computational thinking education, in *Computational Thinking Education*, edited by S. C. Kong and H. Abelson (Springer, Singapore, 2019), 10.1007/978-981-13-6528-7_1.

[22] D. Catlin and J. Woollard, Educational robots and computational thinking, in *Proceedings of 4th International Workshop Teaching Robotics, Teaching with Robotics and 5th International Conference Robotics in Education* (2014), https://www.researchgate.net/publication/264043999_Educational_Robots_and_Computational_Thinking.

[23] T. O. B. Odden, A. Marin, and M. D. Caballero, Thematic analysis of 18 years of physics education research conference proceedings using natural language processing Phys. Rev. Phys. Educ. Res. **16**, 010142 (2020).

[24] AIP Statistical Research Center, Initial Employment—Physics Bachelors and PhDs Classes of 2019 and 2020, https://ww2.aip.org/statistics.

[25] L. Hsu, E. Brewe, T. M. Foster, and K. A. Harper, Resource letter RPS-1: Research in problem solving, Am. J. Phys. **72**, 1147 (2004).

[26] D Fortus, The importance of learning to make assumptions, Sci. Educ. **93**, 86 (2009).

[27] G. Kortemeyer, Could an artificial-intelligence agent pass an introductory physics course?, Phys. Rev. Phys. Educ. Res. **19**, 010132 (2023).

[28] T. O. B. Odden and M. D. Caballero, Computational essays: An avenue for scientific creativity in physics, presented at PER Conf. 2019, Provo, UT, 10.1119/perc.2019.pr.Odden.

[29] S. I. Swaid, Bringing computational thinking to STEM education, Proc. Manuf. **3**, 3657 (2015).

[30] K. Harangus and Z. Kátai, Computational thinking in secondary and higher education, Proc. Manuf. **46**, 615 (2020).

[31] C. Wang, J. Shen, and J. Chao, Integrating computational thinking in STEM education: A literature review, Int. J. Sci. Math. Educ. **20**, 1949 (2022).

[32] J. Malyn-Smith and I. Lee, Application of the occupational analysis of computational thinking enabled STEM professionals as a program assessment tool, J. Comput. Sci. Educ. **3**, 2 (2012).

[33] J. A. Lyon and A. J. Magana, The use of engineering model-building activities to elicit computational thinking: A design-based research study, J. Eng. Educ. **110**, 184 (2021).

[34] Computational modeling in physics with boostrap, https://www.compadre.org/precollege/cmp/, 2018.

[35] R. Vieyra and J. Himmelsbach, Teachers' disciplinary boundedness in the implementation of integrated computational modeling in physics, J. Sci. Educ. Technol. **31**, 153 (2022).
[36] C. M. Orban and R. M. Teeling-Smith, Computational thinking in introductory physics, Phys. Teach. **58**, 247 (2020).
[37] M. D. Caballero, M. A. Kohlmyer, and M. F. Schatz, Implementing and assessing computational modeling in introductory mechanics, Phys. Rev. ST Phys. Educ. Res. **8**, 020106 (2012).
[38] D. P. Weller, Theodore E. Bott, Marcos D. Caballero, and Paul W. Irving, Development and illustration of a framework for computational thinking practices in introductory physics, Phys. Rev. Phys. Educ. Res. **18**, 020106 (2022).
[39] See Supplemental Material at http://link.aps.org/supplemental/10.1103/PhysRevPhysEducRes.20.010128 for all interview questions.
[40] T. O. B. Odden, E. Lockwood, and M. D. Caballero, Physics computational literacy: An exploratory case study using computational essays, Phys. Rev. Phys. Educ. Res., **15**, 020152 (2019).
[41] NVivo (released in March 2020), https://www.qsrinternational.com/nvivo-qualitative-data-analysis-software/home.
[42] J. W. Creswell, *Research Design Qualitative, Quantitative, and Mixed Methods Approaches* (SAGE Publications, Thousand Oaks, CA, 2009), Vol. 20.
[43] B. G. Glaser, The constant comparative method of qualitative analysis, Social problems **12**, 436 (1965).
[44] R. W. Chabay and B. A. Sherwood, Modern mechanics, Am. J. Phys. **72**, 439 (2004).
[45] Association of American Medical Colleges. (n.d.). About the MCAT® exam. Students & residents, https://students-residents.aamc.org/about-mcat-exam/about-mcat-exam.
[46] S. Bocconi, A. Chioccariello, and J. Earp, The Nordic approach to introducing computational thinking and programming in compulsory education, Report prepared for the Nordic@BETT2018 Steering Group, 2018, 10.17471/54007.
[47] Z. Hazari, G. Sonnert, P. M. Sadler, and M.-C. Shanahan, Connecting high school physics experiences, outcome expectations, physics identity, and physics career choice: A gender study, J. Res. Sci. Teach. **47**, 978 (2010).
[48] H. J. Banda and J. Nzabahimana, Effect of integrating physics education technology simulations on students' conceptual understanding in physics: A review of literature, Phys. Rev. Phys. Educ. Res. **17**, 023108 (2021).
[49] C. Vieira, A. J. Magana, A. Roy, and M. L. Falk, Student explanations in the context of computational science and engineering education, Cognit. Instr. **37**, 201 (2019).
[50] A. Backer, Computational physics education with Python, Comput. Sci. Eng. **9**, 30 (2007).
[51] W. Flannery, The coming revolution in physics education, Phys. Teach. **57**, 493 (2019).
[52] T. Young Nicholas, G. Allent, M. Aiken John, R. Henderson, and M. D. Caballero, Identifying features predictive of faculty integrating computation into physics courses, Phys. Rev. Phys. Educ. Res. **15**, 010114 (2019).
[53] A. J. Magana and G. S. Coutinho, Modeling and simulation practices for a computational thinking-enabled engineering workforce, Comput. Appl. Eng. Educ. **25**, 62 (2017).
[54] R. Landau, Computational physics: A better model for physics education?, Comput. Sci. Eng. **8**, 22 (2006),
[55] R. W. Chabay and B. A. Sherwood, Bringing atoms into first-year physics, Am. J. Phys. **67**, 1045 (1999).
[56] R. Chabay and B. Sherwood, Computational physics in the introductory calculus-based course, Am. J. Phys. **76**, 307 (2008).
[57] W. M. MacDonald, E. F. Redish, and J. M. Wilson, The MUPPET Manifesto: The Maryland University Project on Physics and Educational Technology uses microcomputers in the classroom to teach physics, Comput. Phys. **2**, 23 (1988).
[58] A. Pawlak, P. W. Irving, and M. D. Caballero, Learning assistant approaches to teaching computational physics problems in a problem-based learning course, Phys. Rev. Phys. Educ. Res. **16**, 010139 (2020).
[59] C. R. Starr, "I'm Not a Science Nerd!": STEM stereotypes, identity, and motivation among undergraduate women, Psychol. Women Q. **42**, 489 (2018).
[60] H. Shoaib, A. Madamanchi, E. Pienaar *et al.*, "I Think I Am Getting There" Understanding the computational identity of engineering students participating in a computationally intensive thermodynamics course, Biomed. Eng. Educ. **3**, 1 (2023).
[61] J. M. Blaney and J. G. Stout, Examining the relationship between introductory computing course experiences, self-efficacy, and belonging among first-generation college women, in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (2017), pp. 69–74, https://www.researchgate.net/publication/314293657_Examining_the_Relationship_Between_Introductory_Computing_Course_Experiences_Self-Efficacy_and_Belonging_Among_First-Generation_College_Women.