How Computational Physics Students Improve their Computational Literacy

Karl Henrik Fredly,[1] Tor Ole Odden,[1] and Benjamin Zwickl[1,2]

[1] *Center for Computing in Science Education, University of Oslo, 0316 Oslo, Norway*
[2] *School of Physics and Astronomy, Rochester Institute of Technology, 84 Lomb Memorial Drive, Rochester, NY, 14607*

Physics students now use computation from the introductory level until the advanced level, but little is known about how or why they develop their programming skill. We have interviewed computational physics students toward the end of their studies to understand how and why their programming practice has changed. We found that there is a qualitative change in how they write and think about code when they encounter large collaborative coding projects. These projects challenge students with their inherent complexity, pushing them to make their code more readable and sophisticated. The collaborative nature of these projects also play a big part in making students reflect on their choices and learn from each other. Though this is only one example of how to teach computational physics, we discuss implications for teaching from these student experiences.

2024 PERC Proceedings edited by Ryan, Pawl, and Zwolak; Peer-reviewed, doi.org/10.1119/perc.2024.pr.Fredly

I. INTRODUCTION

Computation is important in physics, both for research and for education. It offers great ways for students to learn core concepts such as motion and vectors, and it can offer students greater agency and more exciting problems to explore [1, 2]. Additionally, computation is important to doing physics research and applying physics knowledge to real world problems. Computation is also a complex endeavour, with specialized knowledge and experience being required to implement advanced methods. However, specialized computational knowledge (e.g., vectorization, parallelization) is not present in introductory physics courses, as physics and coding fundamentals take priority [3, 4]. And despite being present in advanced courses, these aspects are so far understudied in the physics education literature. It is important to understand how computation is done and learned at this level, as students need to be well prepared for professional practice, and because programming at this level is hard, qualitatively different and necessary for doing advanced physics [5–7].

The University of Oslo has incorporated computation into nearly all its bachelor's and master's physics courses over many years, and has computational science master's programs specializing in physics and other domains. We are therefore uniquely positioned to study how computation is used and learned at an advanced level, and to hopefully improve instruction at our own and other institutions. Within this context, we study this important topic, and specifically answer:

1. *What motivates computational physics students to improve their computational literacy?*

2. *What resources do computational physics students draw on to improve their computational literacy?*

We will answer these questions by analyzing interviews with computational physics master's students, and comparing their experience with how computation is used in introductory physics, drawing from examples from the literature.

II. THEORETICAL FRAMEWORK: COMPUTATIONAL LITERACY

In order to better describe the role of computation in physics, we will be employing the theoretical framework of Computational Literacy (CL), first formulated by DiSessa [8]. In this framework, computation is described as a literacy, like written or mathematical literacy, with its own representational form and cultural significance and norms. Much like how algebra empowers us to solve and think about problems that would be difficult to wrap our heads around using only words, computation has its own set of affordances. Our analysis will also draw on previous research which has focused on computational literacy in the physics domain [9, 10], and how it is developed in this domain [11]. Computational Literacy consists of three pillars.

The *material* pillar relates to the writing of code, the material representation in this literacy. This pillar consists of knowledge of programming syntax and how the computer works, coding strategies, and beliefs about how code should be. A student with basic literacy in the material pillar would know the basic syntax of a programming language, concepts like loops, conditionals and functions. A student with more advanced literacy in the material pillar could have some more specific skills, like knowing how to use several libraries, vectorization or GPU programming. They could also have more general skills, like reading documentation efficiently and structuring code in a way that is easily expanded and tested.

The *cognitive* pillar relates to applying coding to some domain, in our case physics. Creating or evaluating a physics model fits under this pillar. A student with basic literacy in the cognitive pillar could know how to implement simple domain-relevant numerical methods, and how to tweak input parameters to see how the behavior of a physical system changes. A student with advanced literacy could know how to plan out the solution of a complex problem with code, or how to extract useful information from rich data, what should be computed and visualized.

The *social* pillar relates to how code is used to represent ideas and communicate. We draw from the social pillar when we communicate meaning about code or in code. A student with basic literacy in the social pillar has some notion of creating meaningful names and comments in code, though they don't see the full value of this practice. They also don't know how to structure large codes, or how to collaborate effectively on code. Students require advanced literacy to effectively discuss others' code, and to write code in a way that others will understand.

When we start analyzing real coding activities, we notice that the pillars have some overlap. Consider a student saying to another that a simulated ball doesn't hit the ground due to a loop ending too soon, and that it's probably because of a coding error. All three pillars are clearly intermingled in the activity. The utility then, is not to sort activities into these pillars, but to use these pillars as lenses to understand the computational literacy present in the students. The pillars also feed into each other. If a student feels strongly about improving the readability of their code, they will probably need to improve their material computational literacy in order to do so. And if a student has a very good handle on the material aspects of computation, they will have an easier time finding numerical solutions to physics problems.

We will use this framework to describe why and how computational physics master's students improve their computational literacy. Dynamics like the one just mentioned will be central to our discussions.

III. BACKGROUND

In order to better understand the starting point of the master's students interviewed for this study, we will describe how computation is incorporated into physics courses at the University of Oslo.

The University of Oslo has incorporated computation into its physics program for many years, focusing on computation from the first semester of their bachelor's program, and offering an array of computational science master's programs. Physics students take a course on programming for scientific applications in their very first semester, and go on to use programming in almost every single course they take. Course instructors beyond the first semester can in most cases assume that their students know how to program in Python before taking their course, making incorporation of computation much easier.

In Norway, bachelor's programs span 3 years of coursework, while master's programs consist of one year of coursework followed by one year of thesis work. Around half of all students go on to finish a master's after their bachelor's. Though the students interviewed for this study had varied educational backgrounds, most took a bachelor's in physics at the University of Oslo before starting on their master's in computational physics. The bachelor's program consists of about two thirds physics courses. We mention some of the types of courses most of these computational physics students encountered in their bachelor's and master's studies.

In both the third semester electromagnetism course, and the fourth semester waves and oscillations course, students write computational essays[1]. In these assignments, students freely choose a computational problem relevant to the course. They then formulate a research question, choose a computational method, and present their code and analysis in an essay within a computational notebook. These assignments offer students greater agency and experience presenting and collaborating on code.

Also, the students can take several project-based courses from their fifth semester and onwards. In each course, students typically complete around 3-5 large numerical projects, preferably in groups, instead of taking exams. Many computational physics students take the computational physics course and the applied machine learning course, which are offered at the end of their bachelor's or at the start of their master's. Prior research has covered how project-based courses can help teach expert practices in computational physics [12]. In the results section, we will detail how computational physics masters students experienced taking these courses.

IV. METHOD

We interviewed a total 12 of students enrolled in a computational physics master's program for this study (11 men; 1 woman). Three were just starting the program, 2 were starting their second year, and 7 had just delivered their thesis. The students were recruited by visiting their joint offices and inviting them to participate in the study.

The interviews lasted around an hour each, and were conducted in English or Norwegian. Some data presented were translated and are marked as such. The interview protocol was designed to cover the different pillars of computational literacy. The interview first focused on their thesis work, then their programming practices and views, and finally how courses, fellow students, and other sources had contributed to building these practices and views. During the interview, students also presented and discussed snippets of code relevant to their work. The students did most of the talking, with the interviewer mostly asking follow-up questions or leading the conversation into the next topic, as most of them had a lot to say. The interviews were transcribed and anonymized.

We analyzed the data using thematic analysis [13], with an approach similar to that of Odden et al. [9]. The first author, who also conducted the interviews, took a first pass through the data, noting down elements of interest in regard to the students' computational literacy and learning of computation. For the preliminary analysis presented in this paper, we grouped these elements into themes, and selected the five most important ones to be further analyzed and presented.

V. RESULTS

A. What motivates students to improve their CL

1. Students are motivated by beliefs about code

The student population for this study chose a master's program with a big focus on computation, and thus clearly enjoy programming. The data supports this, as seen with Steven:

> **Steven:** So I was collaborating with another person who, he kind of used coding just to get an answer. While I liked code for what code was. [...] I tried to make it as fast as possible and just as general as possible.

Here we see a student who likes programming for its own sake. For them, code doesn't need to be made nice or fast for some practical reason. They care about optimizing a script for readability and speed, even though it will only be used once. These strong beliefs fit nicely in the material pillar, without relating much to the social or cognitive. Some students however, want to write better code because they want to become good physicists:

> **Levi:** But if science is collaborative, and I really do believe it is, then you shouldn't write ugly code.

Here we see a belief that fits in the social pillar, as Levi sees value in code being nice and readable for the sake of collaboration. The students had varied motivations, some caring more about code for its own sake, others about the collaborative nature of code, and some about code as a tool for physics. These sentiments motivate the students to write good code, and to improve their abilities.

But for all of these students, higher goals often get left behind because of time pressure and wanting to get assignments done. Most students talked about this trade off, even the student from before:

> **Levi:** I feel like I'm always trying to make my code better in that sense, but also trying not to compromise the pace of the project, right?

Many students want to restructure, document and test their code, for the sake of the learning, and for the sake of the doing good work, but they don't feel like they have the time. Additionally, the structure of computational activities in the earlier classes may discourage the use of these practices. For the first few semesters, all they do is write small scripts implementing simple methods. They can make sense of the code just fine, so readability and structure aren't that important. Most courses also don't assess the quality of code, giving yet another reason for students to focus on throwing together something that just works.

But then something comes along and forces these pragmatic students into writing code that actually is planned out, structured, and readable: Big coding projects.

2. Big coding projects challenge the students

In several project-based courses later in the bachelor's level, or at the start of the master's level, students encounter big coding projects. In these projects, students need to implement several complex numerical methods, and apply them to several different systems (e.g., quantum systems with different potentials and dimentionalitites). They benchmark the methods, and plot and analyze the systems in several different ways.

A good example is the second project in the applied machine learning course, where students need to implement neural networks and logistic regression, as well as reusing their code for normal regression, bootstrapping and cross-validation from the first project. These methods all need to fit in the same analysis and work on the same data. In the third project, students can apply such methods to physics problems.

Here is how one student describes the experience of taking a different project based course:

> **Jake:** After the computational physics course I learned that, OK, now that I have projects that are this large, that require this much code and this many files, that I can't keep track of everything in my head of what's going on. *(translated)*

When students first encounter these projects, they have a solid material literacy, they know how to write loops and functions, and they might remember how to make classes. But they don't have the literacy needed to use these tools to simplify complex problems. This is hard, and takes a lot of experience. By the end of the course, several students commented that they still felt they had not mastered the practice. These courses challenge the students, which makes them reconsider their programming practices and seek ways to improve both their cognitive and material skills, as new programming language features and tools (e.g., Classes, GithHub, compilers, IDEs) are required to effectively solve these problems. Another factor that makes students seek to improve are their peers.

3. Peers offer reasons to improve

Students often collaborate on code in project-based courses, which gives them new perspectives on their programming. Students commenting on each other's code is the most straightforward way this happens. They need to use each other's code, so they care about making it usable.

> **Steven:** I was collaborating with two people who were like heavily criticizing me in the beginning for writing bad codes. I tried to like fix that.

But this goes the other way as well. If they want to be helpful, they should write good code:

> **Levi:** I was using other people's code and people were using my code so I really cared about making it effortless. Yeah okay in my thesis not that much because I mean, of course, I try to write things properly, but I'm doing it kind of alone.

Here we see that Levi cared more about writing effortless, usable code in their collaborative coursework than they currently do in their thesis, likely due to missing the motivation from the social aspect of programming. Students want to get better at understanding the code of others, but also to write understandable code, which requires improvement in both the social and material pillars.

B. What resources students draw on to improve their CL

1. Students learn from working with each other

Students telling each other to improve their code can lead to improvement, but so can helping each other:

> **Casey:** When you get to be the one who knows more once in a while, that's also a little nice. Not just that it helps the self-esteem and stuff, but you actually reflect on what you prefer. You turn on a critical sense. *(translated)*

Here, Casey illustrates how the students' perspective on their coding changes from working with others. This then feeds into the material aspects, as the students work together to improve their code:

> **Alex:** I would normally go and ask other students how they solved the exercise. Either for help, if my program doesn't run, or just to see if they had a better solution than me.

Here we see a student working with peers not necessarily to finish the exercise and move on, but to learn and improve. The pragmatic view is thwarted by aspiration, and by the fun and usefulness of sharing ideas. Some students even compete or work together to write the fastest code. The culture and environment promote this, as students are housed in shared office spaces and are encouraged to work together, promoting discussion on code outside group work and courses.

2. Students must go beyond the course material

Course instruction also improves the computational literacy of students, but mostly the cognitive pillar. Lectures mainly cover the methods and how to think about the physics. Assessments also focus on understanding methods and analysis, with little emphasis on the coding. The interviews present the cognitive pillar as the one most reliant on experience. Students describe having to learn how to plan solutions to computational problems by trial and error. This skill is also harder for a peer or grader to appraise, and can't be learned from a textbook. The cognitive pillar can only take students so far, however. Comfort with the material pillar is also required to solve advanced problems, and so students require more resources than just the lectures.

Code examples in the courses vary in quality and breadth, which leads students to find help online, either from videos, AI assistants, official documentation, or forums like Stack Overflow[14]. As one student puts it, they were not told how to write C++, only to write C++. This approach to teaching has some trade-offs. While online material can supplement course instruction on a number of topics, it can sometimes be inefficient and leave students struggling to even get started.

Despite a wealth of online information, students might never learn better ways to code unless someone tells them about them, be that a peer, an instructor, or a colleague at an internship. Many of the students mentioned that colleagues and supervisors from work opportunities helped them learn better ways to work with code. As teaching institutions, we should strive to give all students access to the best advice.

VI. CONCLUSIONS

In this study, we found that big coding projects motivate students to improve their computational literacy. Students can't easily keep track of the logic in these complex projects, and so they learn how to plan and structure their code. Students are also motivated by collaborating with others, as they try to become as good as their peers and to write good code for the sake of the group.

We also found that students improve their programming by learning from their peers. Students ask for help when they are stuck, and share their code when they make something they are proud of. When students compare their code with others', they reflect on their coding choices, such as structure or choice of methods. This study shows the importance of the social element of computation, but also the importance of access to good course materials, feedback and advice.

This study highlights how learning a programming language is only a stepping stone towards the computational literacy required for advanced computational physics. Instructors should note the importance of project-based learning, and the social aspect of coding. The student difficulties described also illustrate the important role course instruction has in scaffolding the learning process.

Future studies should look into the nature of computational literacy in advanced physics. There is still much to study about how experts use computation, and how students should reach this level, especially students who struggle. These students might not persevere as long or get help from their peers, and so some other structures than the ones described here might be needed. Further research on the social aspects of computation is also warranted, as this culture for sharing and helping with code should be a model for all computational physics communities.

ACKNOWLEDGMENTS

This work was funded by the Norwegian Agency for International Cooperation and Quality Enhancement in Higher Education (DIKU) which supports the Center for Computing in Science Education. We thank Sarah McHale and Danny Cabellero for their help and feedback with this project.

[1] T. O. B. Odden, D. W. Silvia, and A. Malthe-Sørenssen, *"Using computational essays to foster disciplinary epistemic agency in undergraduate science,"* Journal of Research in Science Teaching **60**, 937 (2023).

[2] S. Papert, *Mindstorms: Children, Computers, And Powerful Ideas* (Basic Books, 2020).

[3] D. Weintrop, E. Beheshti, M. Horn, K. Orton, K. Jona, L. Trouille, and U. Wilensky, *"Defining Computational Thinking for Mathematics and Science Classrooms,"* Journal of Science Education and Technology **25**, 127 (2016).

[4] D. P. Weller, T. E. Bott, M. D. Caballero, and P. W. Irving, *"Development and illustration of a framework for computational thinking practices in introductory physics,"* Physical Review Physics Education Research **18**, 020106 (2022), publisher: American Physical Society.

[5] G. Miller, *"A scientist's nightmare: Software problem leads to five retractions,"* Science (New York, N.Y.) **314**, 1856 (2007).

[6] G. Balaban, I. Grytten, K. D. Rand, L. Scheffer, and G. K. Sandve, *"Ten simple rules for quick and dirty scientific programming,"* PLOS Computational Biology **17**, e1008549 (2021), publisher: Public Library of Science.

[7] G. K. Sandve, A. Nekrutenko, J. Taylor, and E. Hovig, *"Ten Simple Rules for Reproducible Computational Research,"* PLOS Computational Biology **9**, e1003285 (2013), publisher: Public Library of Science.

[8] A. diSessa, *Changing Minds: Computers, Learning, and Literacy* (The MIT Press, 2000).

[9] T. O. B. Odden, E. Lockwood, and M. D. Caballero, *"Physics computational literacy: An exploratory case study using computational essays,"* Physical Review Physics Education Research **15**, 020152 (2019), publisher: American Physical Society.

[10] T. O. B. Odden and M. D. Caballero, in *The International Handbook of Physics Education Research: Learning Physics*, edited by M. F. Taşar and P. R. L. Heron (AIP Publishing LLCMelville, New York, 2023) pp. 19–1–19–28.

[11] T. O. B. Odden and B. Zwickl, "How Physics Students Develop Disciplinary Computational Literacy," (2024), arXiv:2403.15028 [physics].

[12] C. Burke and T. J. Atherton, *"Developing a project-based computational physics course grounded in expert practice,"* American Journal of Physics **85**, 301 (2017), arXiv:1604.06122 [physics].

[13] V. Braun and V. Clarke, *"Using thematic analysis in psychology,"* Qualitative Research in Psychology **3**, 77 (2006).

[14] "Stack overflow," `stackoverflow.com`, accessed: 2024-07-16.