Coding Expediently: A Computationally Situated Epistemic Game

Austin Anderson, Paige Pressler, and W. Brian Lane
Department of Physics, University of North Florida, 1 UNF Drive, Jacksonville, FL, 32224

As physics educators integrate computational activities, they must attend to the overarching processes that students follow when interacting with computer code. We describe one such process, Coding Expediently, as an epistemic game that students might play with a goal of minimizing the amount of time or keystrokes required to carry out a set of programming steps. Playing this game reduces the time and cognitive load students devote to step-by-step interaction with computer code, leaving more time and cognitive load for bigger-picture sense-making. We describe observations of two students playing this game during think-aloud interviews in which they completed an introductory Python tutorial. These students represent two differing technical backgrounds: rich experience with mathematics and no programming experience, and moderate experience with mathematics and rich programming experience outside of Python. We describe observations of these students playing this epistemic game selectively (even when they are aware of its benefits) and how they play this game in significantly different ways.

2022 PERC Proceedings edited by Frank, Jones, and Ryan; Peer-reviewed, doi.org/10.1119/perc.2022.pr.Anderson

I. INTRODUCTION

Learning activities that integrate computer programming with physics concepts are demonstrably effective at helping students develop both their conceptual understanding of physics material and a transferable computational skill set [1–23]. However, delivering computationally integrated learning experiences means that physics educators must be prepared to train students in basic programming practices (just as we train them in laboratory and analytical practices) [13, 14, 24]. In PER and beyond, much attention has been devoted to how students make sense of a code and use computation as an extension of their thinking [3–5, 8, 15, 20, 23, 25–34]. Here, we focus on tangible steps that students take while interacting with a code during a computational activity. We justify this focus by noting that self-directed programming practices are important transferable skills for student careers [35] and such steps are readily observable in the classroom context. We formalize this focus using epistemic games.

II. CODING EXPEDIENTLY: AN EPISTEMIC GAME

An epistemic game is a locally coherent, goal-oriented set of rules and strategies that guides and limits what knowledge is appropriate for a learner to use in pursuit of creating knowledge or solving a problem [36]. Epistemic games are often represented as flowcharts to capture their procedural nature and the decision-making they require, and can help students bridge their learning between computer science and physics [37]. Epistemic games are described using four components: (1) the game's *epistemic form* (a structure that tells us what the game looks like); (2) a *knowledge base* (the collection of all resources needed for the game); (3) *entry and exit conditions* (a framework that signals when to start and end the game); and (4) a set of *moves* (steps taken during the game).

Here, we propose an epistemic game called Coding Expediently to contrast observations of two students of differing backgrounds learning a new programming language. This game is frequently employed by code developers or computationally oriented researchers to achieve a given outcome using a computer code while minimizing the amount of time or number of keystrokes required. Based on the two cases described here and our insights from using computation in teaching and research, we describe Coding Expediently using Figure 1 and the following components:

1. Epistemic Form: Coding Expediently requires a code to be developed, a goal for the code's output, and multiple paths of modifying the code to reach the goal.
2. Knowledge Base: The student must know the programming language's syntax, the procedures to be carried out, how to translate these procedures into syntax, and how to create a mental representation of what the code is doing or would do with modification.
3. Entry and Exit Conditions: To begin, the student must recognize the existence of multiple paths and activate a motivation for discerning an expedited path. This motivation might include finishing the activity sooner or "getting on with" the main goal of the activity. Ending the game requires recognizing the desired output.
4. Moves: The student must identify the desired output, identify possible paths, evaluate the time or workload for each path, run the code and compare output against the desired output, and revise or continue the path.

Understanding this game can help educators better manage student workload and train students in valuable computational practices for future contexts. For example, Figure 2(a) shows an activity that has the epistemic form of Coding Expediently: The student is presented with a *code to be developed* that approximates e using the first three terms of a series expansion ($e \approx 1 + 1/1! + 1/2!$). The *goal for the output* is for the student to produce a better approximation for e by adding the fourth term ($1/3!$). We consider *two paths* a student could take to modify the code: Manually type new lines of code or copy, paste, and modify the existing lines of code. The *knowledge base* required is a *mental representation* of what the existing lines of code accomplish and what they would accomplish with the addition of a single term. The *entry condition* for this epistemic game is for a student to recognize that copy-paste-modify would be a faster solution method than retyping existing code and choose to pursue the faster method. The *exit condition* is recognizing that the new four-term summation is closer in numerical value to e than the previous. Cues for these entry and exit conditions are provided in the tutorial text. The *moves* in this game are to *identify* the goal of producing a closer approximation for e, *identify* the two paths, *evaluate* the number of keystrokes required to follow each path, and *compare* the new approximation for e with the accepted value for e (presented in the tutorial text).

III. METHODOLOGY

We conducted think-aloud interviews [38] in which student volunteers from a mid-sized regional state university spent 30-40 minutes completing an introductory Python tutorial. The tutorial assumed no prior programming experience and introduced the practices of assigning and reassigning variables, basic computations, printing output to the screen, reading error messages, and fixing errors. The tutorial was presented as a Jupyter notebook [39] hosted on Google Colab [40] to integrate instructions and starter codes, following the recommended practice of minimally working programs [34, 41–43]. The tutorial applied these programming practices by directing the students to set up a calculation of e using the series expansion described above.

We video recorded students' commentary using a camcorder and screen captured their work using Open Broadcaster Software [44]. The screen capture process also recorded audio from the computer's microphone, allowing us to synchronize these recordings and combine them side-by-side in a single video file for review. We asked each student

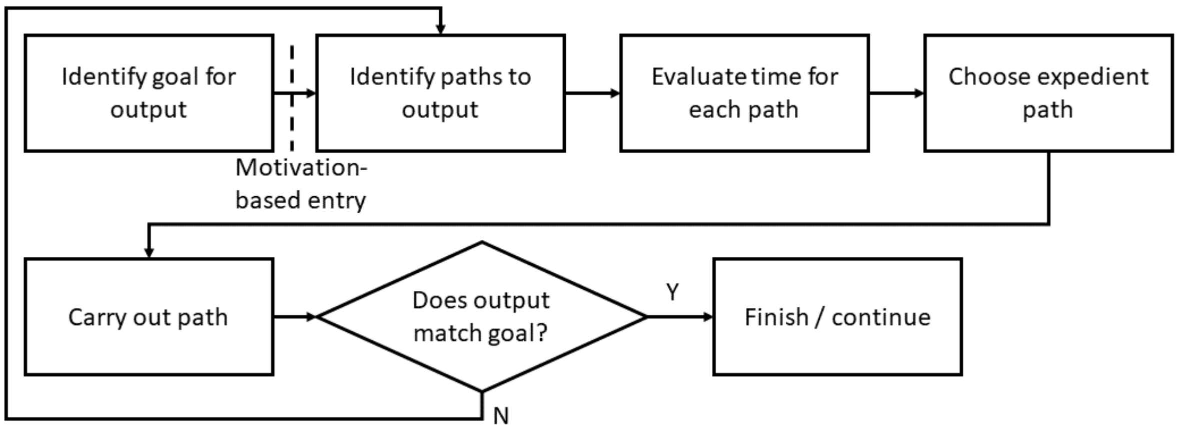


FIG. 1. The Coding Expediently epistemic game.

to think aloud as they worked through the tutorial. The interviewer occasionally needed to prompt the students to resume thinking aloud after a period of silence but otherwise students regularly vocalized their thinking. We supplemented these interviews with a pre- and post-survey conducted on-line. The pre-survey included questions about students' background in math, physics, and programming. The post-survey asked students to reflect on their experience with the tutorial.

During fall 2021, we recruited students using fliers in the physics building and Canvas course announcements in physics classes. Students received a $10 Amazon gift card for their participation. Over the semester, 11 students participated, representing a diversity of prior experiences with programming. As a preliminary report on our findings, we present here two contrasting cases of students with different prior experiences with math and programming.

IV. CONTRASTING CASES

For this preliminary report, we chose two interviews, featuring students Vance and Meghan, as contrasting cases of experience with other programming languages versus experience in calculus and physics. Vance reported taking no prior courses in computer programming or coding, while Meghan reported completing a first-semester programming course in C. Vance reported taking Calculus 3, Calculus-Based Physics 2, and Organic Chemistry, while Meghan reported taking Calculus 2 and Calculus-Based Physics 1.

The first case we discuss presents an opportunity to Code Expediently by using the computer's clipboard (copy-paste) feature to extend an existing code. The second case presents an opportunity to Code Expediently by choosing to modify a minimal number of variable names to resolve an error. In the transcript excerpts that follow, we use parentheses to indicate long pauses with the duration of the pause in seconds, and we use square brackets to indicate the student's interaction with the code as recorded by screen capture at that time.

No explicit instructions to minimize keystrokes, or minimize time on task, were given in the lead-in for these activities. Therefore, we expect to be able to observe students adopting a variety of epistemic games throughout these activities, focusing on Coding Expediently here.

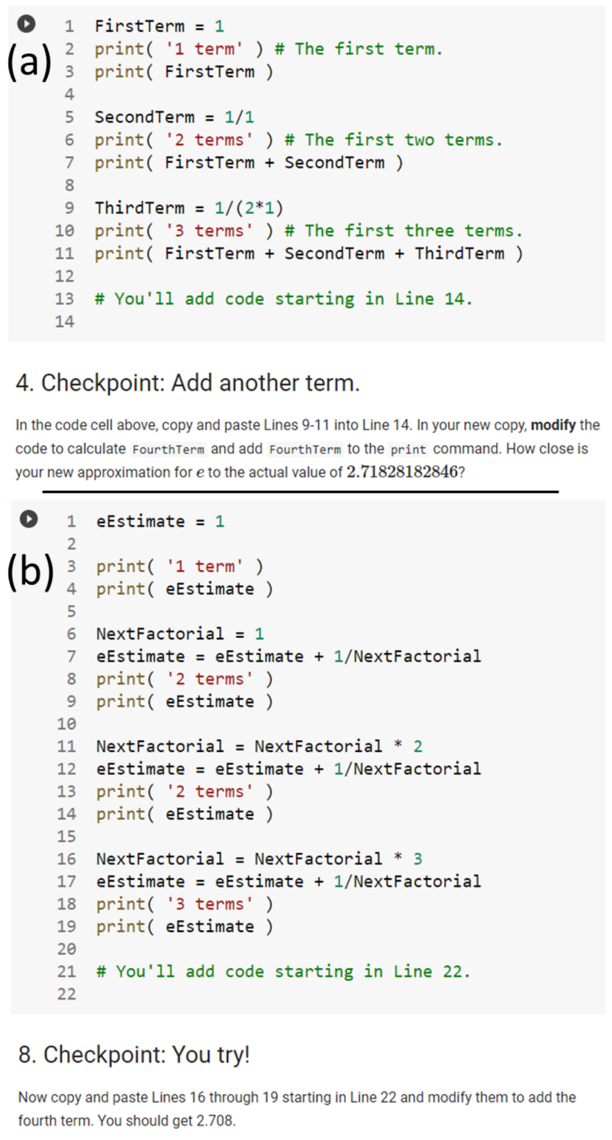
A. Calculating e: Copy-Paste-Modify versus Type Your Work

This portion of the tutorial is shown in Figure 2(a) and was described earlier. First, we examine Meghan's approach: "So I'm just modifying it so that it calculates the fourth term. (6) [copies `ThirdTerm = 1/(2*1)` and pastes into a blank line; changes variable name to `FourthTerm`] Just increasing by one number [adds `3*` to the denominator], because that's what the function does. (15) And then I'm just like copying and pasting the next two lines [copies and pastes two print statements] so that it's all uniform and then I'm just changing it to four terms [changes the first print statement to output `4 terms`] (7) and then in the last line, which is the one where it's going to output the results, I'm just adding fourth term to that [adds `+FourthTerm` to summation] so that's pretty straightforward." We note that Meghan's wording indicates that they are taking a path of actions they consider simple: "I'm *just* modifying... *Just* increasing by one number... *just* like copying and pasting... *just* changing it..." They are not commenting on the ease or difficulty of the mathematical content itself (in fact, other comments indicate they are unsure of what a factorial is), but rather on the mechanical steps of working with the code. These comments indicate that Meghan identifies isolated changes to the existing code that can otherwise be copied and pasted. Assuming they choose this path over the option to type everything out manually, Meghan has passed the entry condition for Coding Expediently. They also use the required knowledge base that predicts what additional commands are required.

We contrast this approach with Vance's: "I'm going to need to continue this. (3) OK, starting on line 14, type in fourth term [begins typing code to construct `FourthTerm` and print `4 terms`] following the format listed. One over three times two times one. (9) Go ahead and print that. (10) And we're going to follow the pattern of leaving a little note after a little pound sign (6) [types a comment to note four terms; types print statement to output new summation]." Vance does not meet the entry requirement for Coding Expediently, proceeding to type out each new line. They proceed to run the code and verify that it produces a closer approximation to e. They then see instructions below the code that reference copying and pasting: "Oh, I could have copied and pasted, great. (3) And here we learn that we should read before we start doing things. Funny, that. Well, that's what I get, I got to type all that. I didn't do that. That's my punishment." They recognize after the fact that there was a more expedient path, and consider the additional keystrokes as a "punishment."

Ironically, in a follow-up task that carries out this same process using progressive variable updates (Figure 2(b)), Vance still opts to type out the new code manually while acknowledging the availability of a more expedient path: "[begins typing out code to produce the next e approximation] I could (3) [continues typing] just copy and paste. But *why would I do that when I can make it all fresh*?" Vance acknowledges the entry condition for Coding Expediently and proceeds to type out as before. Although their tone in the italicized portion of this excerpt sounds sarcastic, we wonder whether this behavior speaks to a different value that, for Vance, supersedes expediency and prompts them to engage in a different epistemic game. We hope to find more examples of this epistemic game in other think-aloud recordings.

In fact, back in the first activity that sets up an approximation for e, Vance's longer path of typing out new code affords them an additional learning opportunity. This episode occurs before Vance realized they could have copied and pasted: "I'm going to screw this up just to investigate what the limits here are. So I'm going to see if there is a variation in the capitalization. I'm not going to capitalize the word term there. [changes `ThirdTerm` to `Thirdterm` in the new print statement] (4) I want to know if something as simple as a capitalization error is going to flub me up. So if it's case sensitive, I have to be very particular about finding out. But if it's not, it's also very good to know." They then run the code and receive an error message, confirming their suspicion that capitalization does matter in Python. This unprompted investigation may be a to-be-characterized epistemic game.



```
1  FirstTerm = 1
2  print( '1 term' ) # The first term.
3  print( FirstTerm )
4
5  SecondTerm = 1/1
6  print( '2 terms' ) # The first two terms.
7  print( FirstTerm + SecondTerm )
8
9  ThirdTerm = 1/(2*1)
10 print( '3 terms' ) # The first three terms.
11 print( FirstTerm + SecondTerm + ThirdTerm )
12
13 # You'll add code starting in Line 14.
14
```

4. Checkpoint: Add another term.

In the code cell above, copy and paste Lines 9-11 into Line 14. In your new copy, **modify** the code to calculate `FourthTerm` and add `FourthTerm` to the `print` command. How close is your new approximation for e to the actual value of 2.71828182846?

```
1  eEstimate = 1
2
3  print( '1 term' )
4  print( eEstimate )
5
6  NextFactorial = 1
7  eEstimate = eEstimate + 1/NextFactorial
8  print( '2 terms' )
9  print( eEstimate )
10
11 NextFactorial = NextFactorial * 2
12 eEstimate = eEstimate + 1/NextFactorial
13 print( '2 terms' )
14 print( eEstimate )
15
16 NextFactorial = NextFactorial * 3
17 eEstimate = eEstimate + 1/NextFactorial
18 print( '3 terms' )
19 print( eEstimate )
20
21 # You'll add code starting in Line 22.
22
```

8. Checkpoint: You try!

Now copy and paste Lines 16 through 19 starting in Line 22 and modify them to add the fourth term. You should get 2.708.

FIG. 2. These portions of the tutorial prompt the student to calculate e using (a) numerical values and (b) variable reassignment.

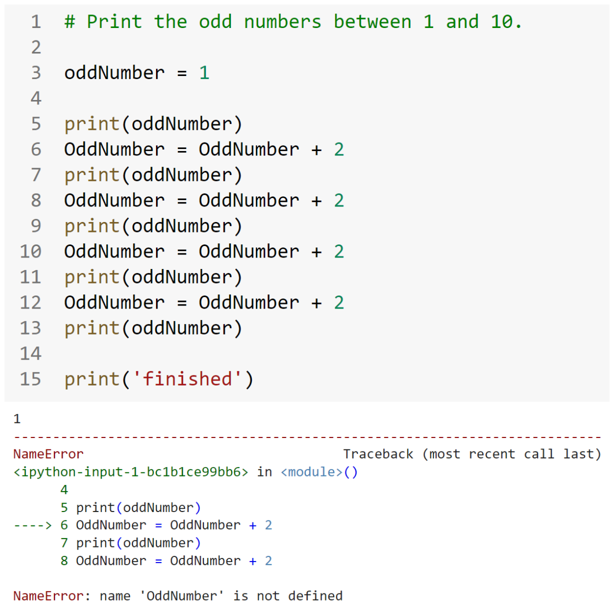
B. Fixing an Error Message: Minimize Keystrokes versus Checking Before Committing

This portion of the tutorial required the students to identify and fix an error in the Python code shown in Figure 3, where uppercase `O`s and lowercase `o`s have been used inconsistently in variable names. The error message points to where this error first occurs and causes the code to halt.

This activity has the *epistemic form* of Coding Expediently: The code needs to be fixed to reach the goal of running without halting. There are at least two paths that a student might follow to fix this error: Replace all the uppercase `O`s with lowercase `o`s, or replace all the lowercase `o`s with uppercase `O`s. This activity requires a *knowledge base* of how capitalization matters in variable names, the procedures of reading error messages and editing an existing code, and a mental representation of why this error message occurs. The *entry condition* is for a student to recognize that multiple paths could lead them to fixing the error. The *exit condition* is for the student to see the code run without error. The *moves* are identifying the goal of resolving the error, identifying the possible paths of capitalization, estimating the time or workload based on the number of uppercase and lowercase instances, and running the code and checking for errors.

Both students quickly identify the inconsistency in capitalization. Meghan explicitly chooses to replace the lower-frequency spelling: "It would just be easier to (4) there's more times where the o is capitalized than where it isn't, so I'm just going to capitalize the o and capitalize every instance where the o is (3) lowercase." Meghan meets the entry condition for Coding Expediently by making a comparison ("It would be easier...") and takes that path. In contrast, Vance chooses to make lowercase the uppercase variable names, missing the entry condition or choosing the less expedient path to prioritize some other motivation. Vance's pathway here might reflect an assumption that the first variable must be setting "the right" spelling.

However, Vance does engage in Coding Expediently in a different way: After identifying the capitalization inconsistency, they test their conclusion that capitalization is responsible for the error message: "[makes the first instance of `OddNumber` lowercase] So I'm going to print that so that if it pops up, I'm not changing any other line now (4) [runs the code] And it's still having issues. (3) [makes the second instance of `OddNumber` lowercase; the error message now "moves down" two lines.] Yeah, so it's the o's. All of the o's change. That's fun. Because if I fix one line it'll just show me the error in the next line, like they're all the same. So that's actually quite nice to figure out what the error is, that if you correct it in one line and print it again because you have it repetitive, that I can get confirmation." Vance *is* Coding Expediently here, by not investing additional time fixing several instances of the same error before confirming this solution.



```
# Print the odd numbers between 1 and 10.

oddNumber = 1

print(oddNumber)
OddNumber = OddNumber + 2
print(oddNumber)
OddNumber = OddNumber + 2
print(oddNumber)
OddNumber = OddNumber + 2
print(oddNumber)
OddNumber = OddNumber + 2
print(oddNumber)

print('finished')
```

```
1
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-bc1b1ce99bb6> in <module>()
      4
      5 print(oddNumber)
----> 6 OddNumber = OddNumber + 2
      7 print(oddNumber)
      8 OddNumber = OddNumber + 2

NameError: name 'OddNumber' is not defined
```

FIG. 3. This portion of the programming tutorial prompts the student to identify a capitalization error.

We recognize that the difference in capitalization frequency in this activity is slight (6 versus 8), and observing this difference in behavior was not an explicit goal for the design of this portion of the tutorial. In future iterations of this study, we will more heavily skew the ratio (1 lowercase original followed by 13 uppercase) to see whether we can observe this difference under more exaggerated conditions.

V. CONCLUSIONS

We have described Coding Expediently as an epistemic game that students sometimes engage in while working with a learning activity situated in a programming context. We have described observations of this game in pairs of contrasting episodes during think-aloud interviews in which students worked through an introductory programming tutorial. In one pair of episodes, we observed one student who explicitly chose a more expedient path (copy-paste-modify) and one student who explicitly chose a longer, more deliberate path (type out original code) even while acknowledging the more expedient path. In the second episode, we observed both students strategizing about expedient behavior that took very different forms. These choices of playing the game (and which moves to take within the game) seem to arise from a difference in outlook toward the tutorial activity, which we interpret as playing different epistemic games. We conclude with a few remarks about this epistemic game.

Coding Expediently could be considered a meta-game, as it is played while pursuing any computational goal (which might invoke its own epistemic game). However, as mentioned earlier, more efficient coding practices reduce the time and cognitive fatigue to work through an activity, allowing learners more time and cognitive load to focus on sense-making in the computational activity. Additionally, Coding Expediently extends beyond the epistemic realm within the student's mind, as it includes interacting with a computer code, which provides a "real-world" check on what students are constructing. Such interactions are not found in mathematical problem-solving, where students can construct any sequence of mathematical claims and steps, with the only feedback coming from their own minds. These differences between Coding Expediently and epistemic games situated in mathematical problem-solving prompt us to consider other epistemic games that might arise in other observations of computational activities. Finally, given the different ways in which our two students engaged in Coding Expediently in the second contrasting case, it will be interesting to observe how groups of students play this epistemic game while collaboratively completing a computational activity.

ACKNOWLEDGMENTS

This work is supported by the University of North Florida Office of Undergraduate Research and the Partnership for In-

tegration of Computation into Undergraduate Physics.

[1] T. J. Bing and E. F. Redish, Symbolic manipulators affect mathematical mindsets, Am. Journ. Phys. **76**, 418 (2008).

[2] M. D. Caballero and S. J. Pollock, A model for incorporating computation without changing the course: An example from middle-division classical mechanics, Am. Journ. Phys. **82**, 231 (2014).

[3] C. E. Wieman, K. K. Perkins, and W. K. Adams, Oersted Medal Lecture 2007: Interactive simulations for teaching physics: What works, what doesn't, and why, Am. Journ. Phys. **76**, 393 (2008).

[4] C. Singh, Interactive learning tutorials on quantum mechanics, Am. Journ. Phys. **76**, 400 (2008).

[5] O. P. Sand, T. O. B. Odden, C. Lindstrøm, and M. Caballero, How computation can facilitate sensemaking about physics: A case study, PERC Proceedings (2018).

[6] A. Buffler, S. Pillay, F. Lubben, and R. Fearick, A model-based view of physics for computational activities in the introductory physics course, Am. Journ. Phys. **76**, 431 (2008).

[7] M. J. Obsniuk, P. Irving, and M. Caballero, A Case Study: Novel Group Interactions through Introductory Computational Physics, PERC Proceedings (2015).

[8] N. Chonacky and D. Winch, Integrating computation into the undergraduate curriculum: A vision and guidelines for future developments, Am. Journ. Phys., **76**, 4 (2008).

[9] D. P. Weller, K. Hinko, and V. Sawtelle, Investigating complementary computational and empirical activities for students learning diffusion, PERC Proceedings (2018).

[10] R.M. Serbanescu, P.J. Kushner, and J.S. Stanley, Putting computation on a par with experiments and theory in the undergraduate physics curriculum, Am. Journ. Phys., **79**, 9 (2011).

[11] R. Chabay and B. Sherwood, Computational physics in the introductory calculus-based course, Am. Journ. Phys. **76**, 307 (2008).

[12] N. T. Young, G. Allen, J. M. Aiken, R. Henderson, and M. D. Caballero, Identifying features predictive of faculty integrating computation into physics courses, Phys. Rev. Phys. Educ. Res. **15**, 010114 (2019).

[13] M.D. Caballero, M. Kohlmyer, and Michael Schatz, Fostering Computational Thinking In Introductory Mechanics, PERC Proceedings (2011).

[14] M. D. Caballero, M. A. Kohlmyer, and M. F. Schatz, Implementing and assessing computational modeling in introductory mechanics, Phys. Rev. S.T. Phys. Educ. Res. **8**, 020106 (2012).

[15] H. A. Dwyer, B. Boe, C. Hill, D. Franklin, and D. Harlow, Computational Thinking for Physics: Programming Models of Physics Phenomenon in Elementary School, PERC Proceedings (2013).

[16] T.O.B. Odden and M.D. Caballero, Computational Essays: An Avenue for Scientific Creativity in Physics, PERC Proceedings (2019).

[17] R.F. Martin, Undergraduate computational physics education: uneven history and promising future, Journ. Phys.: Conf. Ser. **759**, 1 (2016).

[18] Y. Joglekar, G. Vemuri, and A. Gavrin, Normalizing computation through continuous student engagement in the undergraduate physics curriculum, Bulletin of the American Physical Society **65** (2020).

[19] C.J. Burke and T.J. Atherton, Developing a project-based computational physics course grounded in expert practice, Am. Journ. Phys. **85**, 4 (2017).

[20] T.E. Bott, D.P. Weller, M.D. Caballero, and P. Irving, Student-identified themes around computation in high school physics, PERC Proceedings (2019).

[21] N. Hawkins, M.J. Obsniuk, P. Irving, and M.D. Caballero, Examining Thematic Variation in a Phenomenographical Study on Computational Physics, PERC Proceedings (2017).

[22] A. Gavrin, G. Vemuri, and D. Maric, Preliminary efforts to evaluate an initiative introducing computation across the undergraduate physics curriculum, PERC Proceedings (2021).

[23] J. Weber and T. Wilhelm, Conceptual understanding of Newtonian dynamics in a comparative study of computational modeling and video motion analysis, PERC Proceedings (2021).

[24] Z. Rowatt, R. Rosenblatt, and R. Zich, Investigating peer and teaching assistant interactions of physics students working on computational coding, PERC Proceedings (2017).

[25] C. M. Orban and R. M. Teeling-Smith, Computational Thinking in Introductory Physics, Phys. Teach. **58**, 247 (2020).

[26] C. Fracchiolla and M. Meehan, Computational practices in introductory science courses, PERC Proceedings (2021).

[27] C. Wang, J. Shen, and J. Chao, Integrating computational thinking in stem education: A literature review, Int. Journ. Sci. Math. Educ., 1–24 (2021).

[28] B.R. Lunk, Using Conceptual Blending to model how we interpret computational models, PERC Proceedings (2019).

[29] X. Tang, Y, Yin, Q. Lin, R. Hadad, and X. Zhai, Assessing computational thinking: A systematic review of empirical studies, Computers & Education, **148** 103798 (2020).

[30] T. Hsu, S. Chang, and Y. Hung, How to learn and how to teach computational thinking: Suggestions based on a review of the literature, Computers & Education, **126** 296 (2018).

[31] J. Nouri, L. Zhang, L. Mannila, and E. Norén, Development of computational thinking, digital competence and 21st century skills when learning programming in K-9, Education Inquiry, **11** 1 (2020).

[32] D. P. Weller, and T. E. Bott, M. D. Caballero, and P. W. Irving, Developing a learning goal framework for computational thinking in computationally integrated physics classrooms, arXiv 2105.07981 (2021).

[33] D. Weintrop, E. Beheshti, M. Horn, K. Orton, K. Jona, L. Trouille, and U. Wilensky, Defining computational thinking for mathematics and science classrooms, Journ. Sci. Educ. Tech. **25**, 1 (2016).

[34] B.R. Lunk, A Framework for Understanding Physics Students' Computational Modeling Practices., Ph.D. Thesis, North Carolina State University (2012).

[35] P.W. Irving, M.J. Obsniuk, and M.D. Caballero, P3: a practice focused learning environment, Eur. Journ. Phys. **38**, 5 (2017).

[36] J. Tuminaro and E.F. Redish, Elements of a cognitive model of physics problem solving: Epistemic games, Phys. Rev. S.T. Phys. Educ. Res. **3**, 020101 (2007).

[37] R. Taub, M. Armoni, and M. Ben-Ari, Abstraction as a Bridging Concept between Computer Science and Physics, Proceed-

ings of the 9th Workshop in Primary and Secondary Computing Education (2014).

[38] E. Charters, The use of think-aloud methods in qualitative research an introduction to think-aloud methods, Brock Educ. Journ. **12**, 2 (2003).

[39] A. Cardoso, J. Leitão, and C. Teixeira, Using the Jupyter notebook as a tool to support the teaching and learning processes in engineering courses, International Conference on Interactive Collaborative Learning (2018).

[40] https://colab.research.google.com/notebooks/intro.ipynb

[41] D.P. Oleynik and P. Irving, Scientific Practices in Minimally Working Programs, PERC Proceedings (2019).

[42] S. Weatherford, Student Use of Physics to Make Sense of Incomplete but Functional VPython Programs in a Lab Setting, Ph.D. Thesis, North Carolina State University (2011).

[43] S. Weatherford and R. Chabay, Student predictions of functional but incomplete example programs in introductory calculus-based physics, PERC Proceedings (2012).

[44] https://obsproject.com/