# Using Mathematical Knowledge When Learning to Program: The Case of Jack

Zackery Reed
Embry-Riddle Aeronautical University Worldwide

Elise Lockwood
Oregon State University

*Students seeking careers in STEM will increasingly be required to develop proficiency with computational tools. As such, many students will be at least incentivized, if not required, to learn to program early on in their undergraduate careers. We hypothesize that, though many opportunities exist in which students can learn programming skills, providing undergraduate students with opportunities to program in the context of mathematics courses will be advantageous for students. We report on interviews conducted with students as they learned to program in the MATLAB language, undergraduate mathematics problems. We discuss how a student's mathematical knowledge mediated his programming practices.*

*Keywords:* Undergraduate Mathematics, Programming, Computation, Digital Tools

There have been several recent calls for investigations into the role of digital experiences, including programming and computational activity, at the undergraduate level. In a recent special issue in the International Journal of Research in Undergraduate Mathematics Education, guest editors noted that "further research on digital experiences at the university level is definitely needed" (Geraniou et al., 2024, p. 688). They included "computational thinking and programming instructional design and policies" (p. 688) in university mathematics in a list of topics to be investigated. In addition, Lockwood and Morken (2021) explicitly call for "more studies in undergraduate mathematics education that explicitly explore relationships between computing and the teaching and learning of mathematics" (p. 404), saying that "much more needs to be investigated about the role of computing within RUME, including identifying and exploring potential benefits, affordances, challenges, and problematic issues" (p. 405).

In response to these calls, we conducted a teaching experiment (Steffe & Thompson, 2000) in which the solving of undergraduate mathematics problems was leveraged as a context in which the students might learn to program. We briefly report on initial instances of a student leveraging mathematical knowledge to mediate his programming activity. That is, we are interested in addressing the following research question: *In what ways might students' mathematical knowledge mediate their early engagement in machine-based computing?*

## Literature Review

In mathematics education, there is a rich history of exploring the role of the computer in supporting students' learning. Papert (1980) argued that the computer provided opportunities for enriching students' mathematical thinking and activity. Many studies have featured students' use of programming to solve problems in various mathematical contexts including elementary topics (e.g., Benton et al., 2018; DeJarnette, 2019), and specific topics in undergraduate settings (e.g., Buteau et al., 2020; Lockwood & De Chenne, 2020). Our work continues in this vein, focusing on the domain of first year undergraduate mathematics.

Researchers have also focused on the intersection of computational thinking and domain-specific thinking. For instance, multiple studies (Lockwood & De Chenne, 2020; 2021; Medova and Certkova, 2021) reported on affordances of writing and running code for students' combinatorial thinking. In line with these studies, there is a growing literature base exploring

computing and computer programming in STEM and mathematics education (e.g., Caballero et al., 2019; Weintrop et al., 2016; Benton et al., 2018; Sinclair & Patterson, 2018).

We situate our study within this body of work and specifically examine novice coders' experiences learning to code while solving problems in mathematics. That is, "programming" as a practice is applicable to a wide variety of domains beyond computation in STEM fields, and one can learn to "program" in a variety of contexts. We are primarily interested in STEM-intending students initially learning to program in the context of mathematical problem solving. We hypothesize that such contexts can be leveraged effectively to provide students with an increasingly important skillset for the modern workforce.

## Theoretical Framing

Our research question focuses on the interactions between students' mathematical knowledge and their engagement in the activity of machine-based computing (Lockwood & Morken, 2021), particularly as they learn to code for the first time. As such, we theoretically ground both the activity of programming and also mathematical knowledge.

### Machine-based Computing

While framings of computer-adjacent activities such as programming, computing, and algorithmic thinking abound, we are primarily interested in early coders' experiences and activity when learning to utilize computational software to solve math and science problems. We thus focus our account of programming within the broader frame of machine-based computing. Lockwood and Morken (2021) characterize machine-based computing as "…the practice of developing and precisely articulating algorithms that may be run on a machine" (Lockwood & Morken, 2021, p. 405). Programming is an example of machine-based computing as it involves writing structured syntax that makes calls to underlying algorithms (e.g. via variables and functions) to be run in sequence through an interpreter to enact a computation and produce a result.

Importantly for our purposes, the activity of programming involves: a) writing, organizing and structuring specific syntax to be read by the interpreter, and b) making sense of the products of the computations in reference to the task or problem to be solved. These components align with Weintrop et al.'s (2016) characterization of programming, a computational thinking practice that "consists of understanding and modifying programs written by others, as well as composing new programs" (p. 139). We view the "understanding", "modifying" and "composing" aspects of Weintrop et al's characterization as features of programming on which we wish to focus.

### Mathematical Knowledge

We employ a radical constructivist (Glasersfeld, 1995; Piaget, 2013) perspective to characterize students' understandings and infer aspects of their cognition by observing their utterances and mathematical activity. We conceptualize students' development of understandings in terms of their assimilations to schemes (Thompson et al., 2014) and draw from the fundamental constructs of accommodation and abstraction as the underlying mechanisms of learning and knowing. While we only have space to report on broad themes observed across many instances of students' uses of programming to solve mathematics problems, our built models (see the Methodology section) of the students' mathematical understandings that inform these broader themes are rooted in the fundamental constructs of radical constructivism.

Relevant to our analyses are Thompson and colleagues' (2014) notion of meaning and understanding. An understanding is a "cognitive state resulting from an assimilation" and a

thinker's associated meaning is the "space of implications existing at the moment of understanding" (p. 13). Importantly, as we discuss in the Methods section below, the students were leveraging programming to solve problems in mathematics content for which they had already received instruction. As such, the students were drawing from "stable" mathematical meanings (p. 13), that is, meanings for which the related understanding resulted from assimilation to a scheme. This contrasts "in the moment" understandings (p. 13) which still result from assimilation but may be fleeting because of memory constraints or are only momentarily attained in the process of accommodation. Thus, we both examine the salient aspects of the students' stable mathematical understandings being drawn from in the moments of programming, and we also note inferable features of the students' in-the-moment understandings of the logical mechanisms of computation and the workings of the computational interpreter that produced the outputs they observed.

## Methods

We employed Steffe and Thompson's (2000) teaching experiment methodology to examine salient aspects of the students' learning of programming. For this report we focus on the results of individual teaching experiments with students enrolled in mathematics courses at a university in the United States. The teaching experiments entailed 90-minute sessions in which a student utilized MATLAB (Mathworks, 2024) live scripts[1] to solve mathematics problems.

Because of spacing constraints, we report on multiple interviews with a single student, Jack (pseudonym). Jack was in an engineering program and had previously taken the calculus sequence. The selection criterion for the interviews was that the students did not have (or had little) prior experience with programming and were willing to discuss their thoughts as they engaged with the live script.

The interviews took place via Zoom. Participants shared their screens while working in the live script, and the sessions were recorded using the built-in recording feature on Zoom. Participants read (within the livescript) provided textual, diagrammatic, and applet-based teaching elements to learn the syntax, structure, and formatting of code to perform computations in MATLAB. Throughout each live script were frequent goal-orienting prompts to enter code in cells to practice running code to solve problems. Students were free to ask the interviewer clarifying questions and were often prompted to anticipate what they expected the code to produce before running the MATLAB interpreter.

In line with the "responsive and intuitive interaction[s]" (Steffe & Thompson, 2000) common to teaching experiments, the interviewer frequently asked hypothesis-confirming questions while and after students produced and ran code, thusly exploring the participants' reasoning through interaction in the sessions. If the students, after long periods of trial and error, professed to be stuck, the interviewer would aid by providing a hint or parts of the code or syntax.

In retrospective conceptual analysis (Steffe & Thompson, 2000; Thompson, 2008), the research team reviewed the records to build models of the students' thinking using theoretical constructs to explain the observed mathematical activity. The researchers drew from foundational constructs of radical constructivism to infer features of the students' mathematical knowledge being activated and drawn from as they programmed. For this report, the research

[1] Many computational environments have accompanied "notebook" features where code can be structured and formatted alongside text and figures, can be organized into sections, and the output of the code can be viewed within the notebook. Jupyter notebooks are the well-known notebooks for Python. MATLAB's version of this is called a "live script".

team attended to Jack's evolving programming activity, searching for instances where his mathematical knowledge mediated his writing, running, and reviewing of code based on outputs given within the live script. We constructed narratives of episodes in which there were clear or important interactions between Jack's mathematical activity and his programming activity. The episode reported on is exemplary of larger themes taken from the data narratives.

## Results

We present an episode featuring Jack's construction of for loops while solving kinematics-related problems. This episode features both smaller-scale and larger-scale instances of Jack' use of mathematical knowledge while engaged in programming activity.

### "Distance is velocity times time"

In a multi-session activity centered on loops, Jack was tasked with approximating the distance traveled by car when given velocity and timestamp[2] measurements throughout a car's journey. The velocity information was stored in a 1x8 array called "velocities" and the timestamps corresponding to the velocities were stored in a 1x8 array called "times".

The two tasks in this section were to: 1) create a "distances" array with a for loop and use it to approximate the length of the windy road, and 2) create an "accelerations" array giving the average acceleration of the car over each time interval of travel.

Figure 1 gives Jack's first attempted loop[3] to create and use the distances array:

```
velocities=[13 9 10 4.5 2.2 13 6.7 5.4]
times=[0 120 375 447 807 867 959 1079]
%v=d/t
%d=vt
distances=[]
for i=1:8
    distances=velocities.*times
end
sum(distances)
```

```
distances = []
   ⋮
distances = 10^4 × 0 0.1080 0.3750 0.2011 ...
   ⋮
ans = 3.2140 e + 04
```

*Figure 1. Jack's initial loop to produce a distances array.*

Jack noted that this didn't seem correct because the loop gave the same output each time. He then was prompted to explain the output and said the following:

> *J:* So we're doing 13*0, 9*120, 10*375 … so on and so forth. So it's outputting that [i.e. the array output is performing those multiplications.] and then sum(distances) is adding that up.

Jack then noted that the loop was not needed to create the distances array via the calculation distances=velocities.*times, and that the loop was simply performing that calculation multiple times over. Jack then spent some time attempting to alter his code to progressively construct the distances array within the loop, eventually attempting the following:

[2] By "timestamp" we mean the time values at which the corresponding velocity measurements were observed.

[3] In the prior live script, Jack had performed arithmetic on arrays, including using the element-to-element array operation ".*", which multiplied the entries of arrays that share indices.


*Figure 2. Jack's second attempt creating the distances array.*

Jack quickly abandoned this approach. While moving his cursor progressively over the entries in the "times" array he described that for each index i, the loop "… multiplied [the velocities array] by [the time at index] 1, 2, 3, 4, … across times.". We note that Jack was interpreting the structure of the given outputs MATLAB by performing mental arithmetic. Also, implicit in this process was Jack drawing from an in-the-moment understanding of the ordered operations imposed by the MATLAB interpreter that would produce the given outputs. Jack then leveraged this in-the-moment understanding to re-structure his code in a way that reflects the ordered operations to produce the outputs he was expecting.

After another similar attempt, Jack articulated that his loop needed to, at each step, multiply the numbers in the velocities and times arrays that shared an index. After receiving help recalling previously learned syntax, Jack produced a for loop that achieved his stated goals. Importantly, this approach does not accurately approximate the length of the windy road, which was the goal of the task. Jack noted that his results did not match the provided final answer, but he did not act on this discrepancy nor attempt to revisit his code, being confident that the code had achieved his intended calculation. The interviewer chose to move on to the accelerations portion of the task, waiting to see if this discrepancy would surface again.

Jack was then asked to summarize his work on the distance approximation task and write down a formula that would calculate the distance for an arbitrary velocity v and time data t. Jack wrote "D=sum[v1*t1+v2*t2+…]" and said:

> *J:* Distance equals the sum of velocity 1 times time 1 plus velocity 2 times time 2 and so on and so forth … through the whole data set.

We note now (and will elaborate later) that Jack's stated distance calculation matches the calculations that Jack intended to (and eventually did) create with a loop. Jack's distance approximation entailed multiplying the velocity data by their corresponding timestamps, expanding the standard model $d = v * t$ to be enacted with velocity-timestamp pairings.

**"Acceleration is velocity divided by time"**

Moving on, Jack employed a very similar strategy to generate an array that would calculate the approximate accelerations of the car over the intervals of time, saying "Same thing as before … just tit for tat position 1 divided by position 1 in the array, and then 2 and 3 and so forth." Running the interpreter, Jack realized that there was an error somewhere in his code (Figure 3), as the first entry in the accelerations array was Inf (for "infinity").

> *J:* That can't be right … So we have 0 on the first value of "times", so it should be undefined but we're getting infinity for some reason? … So, I mean that's our starting point so our acceleration is 0, I mean but we have a starting velocity … so really we shouldn't be counting that first value I'm thinking. … So I mean acceleration is the velocity minus the original velocity over the delta of time … if we're starting at 13,

that's our velocity original, and then t is still just 0 … so a starting acceleration of 0 so that makes sense. … we're staying at a constant speed.

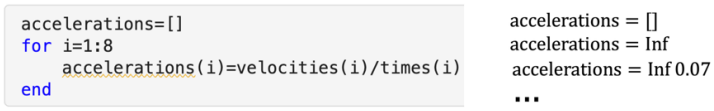


*Figure 3. Jack's first attempt creating an accelerations array.*

Here, we see Jack attempting to justify the "Inf" result by conceptualizing the start point of the data, as if the car had started without acceleration. Jack had suggested that we could ignore the first entry in accelerations, and the entries afterward would capture the "time delta" accurately and so the rest of the array was correct. Following up, the interviewer asked:

*I:* Now you just mentioned that for acceleration you want the delta of the velocities over the delta of the times, and so I'm wondering if in your mind the deltas of the velocities is just the same as the velocity itself in the time slot.
*B:* Yeah I mean acceleration is just velocity over time at that point in time. I guess we could put it that way.

After some time exploring what he meant by "delta" and returning to some of the calculations to examine whether his code was measuring the "delta"s, Jack eventually produced the following loop to calculate the accelerations.


*Figure 4. Jack's final accelerations loop*

We see two different and important instances of Jack's mathematical knowledge mediating his programming practice. First, as Jack built the distances array, he was able to envision element-to-element multiplication operations occurring in multiple structural configurations and make inferences about the MATLAB interpreter and the logic behind the outputs he was seeing. Jack was not only able to parse when an array resulted from one-to-one multiplication across two arrays but also was able to infer the impact of his syntax on the order imposed by the loop.

On a larger scale, Jack's first chosen model for approximating traveled distance was an expansion of the basic model d=v*t to be enacted on each velocity-time pairing in the dataset. This expansion was similar to those of problems in prior lessons of the teaching experiment, where a major theme was the gained ability from MATLAB to make many calculations at once. Supported by Jack's multiple statements such as "distance is velocity times time" and "acceleration is velocity divided by time" and his work on other similar modeling problems, we infer that this notion of making many calculations at once was an underlying goal taken on by Jack that mediated not only his writing of the code, but his interpretation of the validity of the code output. This differs from other possible goals he might have taken on, such as conceptualizing the construction of quantities (Thompson, 1990) necessary for the solution of the problem and then attempting to enact the quantitative operations programmatically, Jack was primarily concerned with making repeated calculations in the form of loops or array arithmetic.

Moreover, Jack required explicit guidance from the interviewer to become aware that his acceleration loop captured point-by-point division but did not capture his stated "delta"s necessary for typical rate of change formulas.

## Discussion and Conclusion

There are two primary takeaways from these results and analyses, both related to the ways that Jack's mathematical understandings mediated his programming activity. The first is that, in these early instances of coding, Jack was developing and leveraging in-the-moment understandings how the program interpreter read his written code. That is, particularly apparent in his for loop refinement, Jack was generating and testing hypotheses about the relationships between their structure of the code and the structure of the resulting output. More than just parsing syntax and debugging errors, there was a cognitive element of inference on Jack's part where he made conjectures about how the code interpreter must be using his written code to produce outputs structured in the ways he observed.

Unlike other contexts in which the results of coding might be interpreted, to parse and interpret the structure of his outputs Jack was engaging in mathematical thinking not unlike the thinking involved in other tasks in the undergraduate curriculum. Students attempting to parse, understand, and leverage formulas such as the distance formula $\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$ or sums such as Riemann Sums, partial sums of series, or infinite series similarly have to generate an understanding of the structure and mechanics of the formulas, but also develop an understanding of the appropriate and expected result of the calculation. Unlike these more traditional instances of envisioned sequences of operations, in computational software the results of many sub-stages of the process are viewable, interpretable, and thus able to provide more figurative material from which a student might mediate their programming or mathematical practice.

Contrastingly, the instances reported on in this study were of basic arithmetic operations. One might imagine similar tasks where the sub-stages are themselves complex and would not bring the same affordance to a novice programmer. Finally, these episodes also highlight potentially important considerations for task design in math-based introductions to programming and computation. Jack's successful structuring and writing of the loops were in some ways independent of the problem's quantitative reasoning requirement for the successful approximation of distance and accelerations. That is, the programming-specific goal of the task was to give students opportunities to flexibly navigate the subtleties of loop construction to produce variables of different types. We saw that Jack was able to achieve this goal even though he did not attend to the quantities in a way that would accurately measure the desired approximation. Rather than making recommendations one way or another, we simply note that future research should further investigate the complexities of the requirements on students to leverage particular understandings or ways of thinking as a precursor to successful program-writing. We consider that one eventual goal would be for students to develop enough flexibility and familiarity with coding to then enable computation to be leveraged as a tool for effective quantification and mathematical reasoning.

## Acknowledgements

This research is based upon work partially supported by the National Science foundation – DUE #2055590.

## References


Benton, L., Saunders, P., Kalas, I., Hoyles, C., & Noss, R. (2018). Designing for learning mathematics through programming: A case study of pupils engaging with place value. *International Journal of Child-Computer Interaction, 16*, 68–76.

Buteau, C., Gueudet, G., Muller, E., Mgombelo, J., & Sacristán, A. (2020). University students turning computer programming into an instrument for 'authentic' mathematical work. *International Journal of Mathematical Education in Science and Technology, 51*(7), 1020–1041.

Caballero, M., Chonacky, N., Engelhardt, L., Hilborn, R., del Puerto, M., & Roos, K. (2019). PICUP: A community of teachers integrating computation into undergraduate physics courses. *The Physics Teacher, 57*(6), 397–399.

DeJarnette, A.F. Students' Challenges with Symbols and Diagrams when Using a Programming Environment in Mathematics. *Digital Experiences in Mathematics Education 5*, 36–58 (2019). https://doi.org/10.1007/s40751-018-0044-5

Geraniou, E., Faggiano, E., Medová, J. et al. Guest Editorial for Special Issue "Digital Experiences in University Mathematics Education: Advances and Expectations". *International Journal of Research in Undergraduate Mathematics Education 10*, 683–689 (2024). https://doi.org/10.1007/s40753-024-00260-4

Glasersfeld, E. v. (1995). *Radical constructivism: A way of knowing and learning.* London: Falmer Press.

Lockwood, E., Mørken, K. (2021). A Call for Research that Explores Relationships between Computing and Mathematical Thinking and Activity in RUME. *International Journal of Research in Undergraduate Mathematics Education 7*, 404–416 https://doi.org/10.1007/s40753-020-00129-2

Lockwood, E., & De Chenne, A. (2020). Enriching students' combinatorial reasoning through the use of loops and conditional statements in Python. *International Journal of Research in Undergraduate Mathematics Education, 6*(3), 303–346.

Lockwood, E., & De Chenne, A. (2021). Reinforcing key combinatorial ideas in a computational setting: A case of encoding outcomes in computer programming. *The Journal of Mathematical Behavior, 62*, (#100857).

The MathWorks Inc. (2024). MATLAB version: 24.2.0.2740171 (R2024b), Natick, Massachusetts: The MathWorks Inc. https://www.mathworks.com

Medova, J. & Ceretkova, S. (2021). Relation between algorithmic and combinatorial thinking of undergraduate students of applied informatics. Paper presented at the *14th International Congress on Mathematics Education*. Shanghai, China (remote presentation).

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas.* Basic Books.

Piaget, J. (2013). *Principles of Genetic Epistemology: Selected Works (Vol. 7).* Routledge.

Sinclair, N., & Patterson, M. (2018). The dynamic geometrisation of computer programming. *Mathematical Thinking and Learning, 20*(1), 54–74.

Steffe, L. P., & Thompson, P. W. (2000). Teaching experiment methodology: Underlying principles and essential elements. *In R. Lesh & A. E. Kelly (Eds.), Research design in mathematics and science education* (pp. 267–307). Mahwah: Lawrence Erlbaum Associates.

Thompson, P. W. (1990). *A theoretical model of quantity-based reasoning in arithmetic and algebraic.* Center for Research in Mathematics & Science Education: San Diego State University.

Thompson, P. W. (2008). Conceptual analysis of mathematical ideas: Some spadework at the foundations of mathematics education. *Proceedings of the annual meeting of the International Group for the Psychology of Mathematics Education* (Vol. 1, pp. 31-49).

Thompson, P. W, Carlson, M. P., Byerley, C., & Hatfield, N. (2014). Schemes for thinking with magnitudes: A hypothesis about foundational reasoning abilities in algebra. *In K. C. Moore, L. P. Steffe & L. L. Hatfield (Eds.), Epistemic algebra students: Emerging models of students' algebraic knowing. WISDOMe Monographs* (Vol. 4, pp. 1-24). Laramie, WY: University of Wyoming.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). *Defining computational thinking for mathematics and science classrooms. Journal of Science Education and Technology, 25*(1), 127–147.