Mathematical Modeling with R: Embedding Computational Thinking into High School Math Classes

By Kenia Wiedemann, *The Concord Consortium*, Jie Chao, *The Concord Consortium*, Benjamin Galluzzo, *Clarkson University* and Eric Simoneau, *Boston Latin School*

Mathematical modeling is routinely used to represent, analyze, and simulate natural and human-made systems. Job ads frequently require computational thinking skills, and new positions open practically daily. However, not enough people apply for these jobs. Many students graduating from high school have no experience in computer science, believing that because they are not experienced programmers, they are not cut out for jobs mentioning terms like machine learning or big data. A new initiative to eliminate the misconception that to think computationally is a synonym to programming proposes embedding computational thinking into curricular subjects. We discuss one such effort and its encouraging results from a curriculum implementation in a U.S. high school.

INTRODUCTION

Numbers and graphs from mathematical models make the headlines daily, banks fight over potential new clients advertising different interest rates, APRs, and other confusing financial terms. It may explain why nine out of ten high schoolers' parents say they would like their children to have access to computer science (CS) courses [5,12]. The good news? The number of schools offering CS in the U.S. is on the rise, and the number of states adopting policies to promote K-12 CS education is also increasing [13]. However, it is estimated that only a little over a third of public high schools in the U.S. currently offer CS classes and only sixteen states have adopted a policy to give all high school students access to CS courses [12].

Occupations under the computer and information technology umbrella described by the Bureau of Labor Statistics (BLS) have a projected growth that goes from 5% to a whopping 32% by 2028 (12% on average) [5] with one notable exception: computer programmers, whose jobs are expected to decrease 7% by 2028 (Figure 1). To quote directly from reference [3], "*computer programming can be done from anywhere in the world, so companies sometimes hire programmers in countries where wages are lower*," strongly suggesting that learning how to program a computer is an important skill to get someone inserted in market of computer and information technology, but it is not the whole story. Newcomers must bring a differential, something in addition to coding skills. These new analysts, developers, system architects, and alike need to demonstrate they can think out of the box and think computationally.

One strategy that can help to build a bridge across this gap between computer and information tech professional supply and demand is to engage students in computational thinking (CT), a concept that predates coding. It is the thought process involved in formulating problems and designing solutions that can be executed by information-processing agents [16]. CT skills are naturally exercised during the process of using mathematical modeling to find solutions to often ill-formulated real-life problems. Embedding CT into mandatory curricular subjects, other than explicit CS classes, extends the reach of CT as well as CS to student populations that otherwise would not have access to it. Not surprisingly, developing computational thinking across all disciplines and educational levels has become a priority for scholars and international agencies that make a call to integrate these concepts across the K-12 and undergraduate curricula.

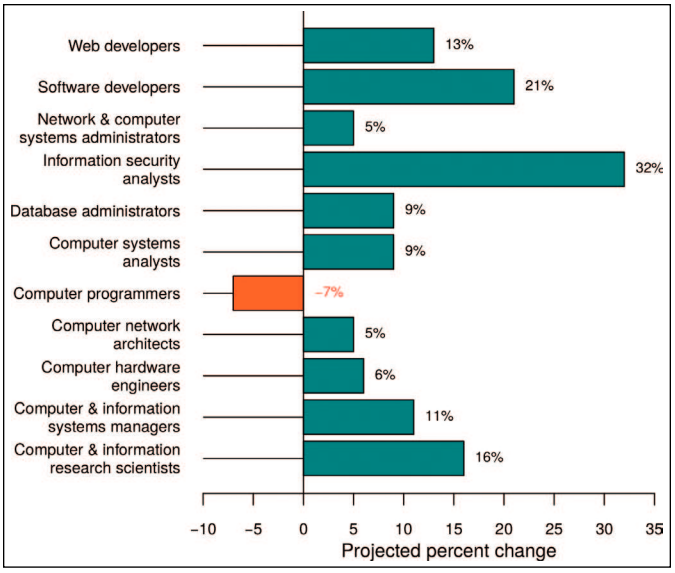

Figure 1. Projected change in employment from 2018 to 2028. Almost all occupations under the computer and information technology umbrella described by the BLS have projected growth for the next few years (with the exception of computer programmers). Yet, many companies report difficulties finding these skilled workers. Exercising computational thinking may help to build a bridge across this gap between computer and information tech professional CS skills' supply and demand.

The Computing with R for Mathematical Modeling (CodeR-4MATH) project, funded by the U.S. National Science Foundation (NSF), has been working to create a collection of learning, assessment, and tutoring resources. Throughout the CodeR-4MATH instructional modules, students learn how to tackle real-world problems using mathematical modeling, with programming language (R) as the modeling environment. During the process, students naturally exercise computational thinking skills while learning basic concepts of computer programming. Developed and supported by the R Foundation for Statistical Computing, R is both a programming language and an open-source environment for statistical analysis, and a favorite among statisticians and data scientists. We developed a series of learning tasks that encourage students to exercise computational thinking, working on topics that are relevant and relatable to them.

The purpose of this research is to design and test instructional modules that can help high school math teachers to embed computational thinking into their classrooms. The CodeR-4MATH modules are designed to allow students to learn how to tackle real-world problems using mathematical modeling, while having a programming language as the modeling environment. In this study we investigate why, and to what extent, engagement in the project activities by high school students enrolled in regular math courses contributes to reshaping their level of interest in further exploring computing related learning opportunities in the future. Notably, student performance demonstrates increased competency in mathematical modeling and computational thinking.

THE INSTRUCTIONAL MODEL

The curriculum modules were developed to engage students in mathematical modeling, using their knowledge of math concepts to solve real-life problems, using two main pedagogical approaches: *context-based* [e.g., 7,11] and *faded scaffolding* [e.g., 4,14, and references therein]. In a context-based approach, everything is taught in the context of the initial open-ended problem. Students learn new concepts and tools only when they need to use them, giving them the motivation to learn something new. Scaffolding is targeted assistance provided to students (either by their teacher or by prompts and hints in self-paced tutorials, for example) as they perform a task. Faded scaffolding can be understood simply as giving support to students when the assistance is needed and removing it when it is no longer needed, giving students the chance to exercise newly acquired skills to solve more complex problems.

The iterative cycle of mathematical modeling involves identifying and selecting parameters to represent a situation, choosing mathematical representations to define those parameters and their relationships, performing mathematical operations to draw conclusions, interpreting and validating the findings against the situation, and iteratively improving the model (Figure 2). Studies have shown that computing activities helped students develop a deeper understanding in a variety of mathematics domains [e.g., 1,9]. When dealing with a problem (real-world or otherwise) to be solved mathematically, it is certain that one can tackle the problem using any media. The ability to teach our understanding of the problem to a computer is referred here as computerizing the problem (cycle 3b in Figure 2). If we don't fully understand part of the system we are trying to describe, the computational results can (and often do) provide us with a useful test for our initial knowledge and hypotheses, or rather what the mathematical results are, either expected or not, providing insights on how to further refine our algorithm.

During the classroom activities, we invited students to analyze and solve problems that are relatively open-ended by design, to encourage students to brainstorm, make assumptions, create algorithms, and hypothesize possible outcomes. Students can then translate their algorithms into a computer code, creating a mathematical model to test their hypotheses for many different scenarios. Based on the model outputs, students

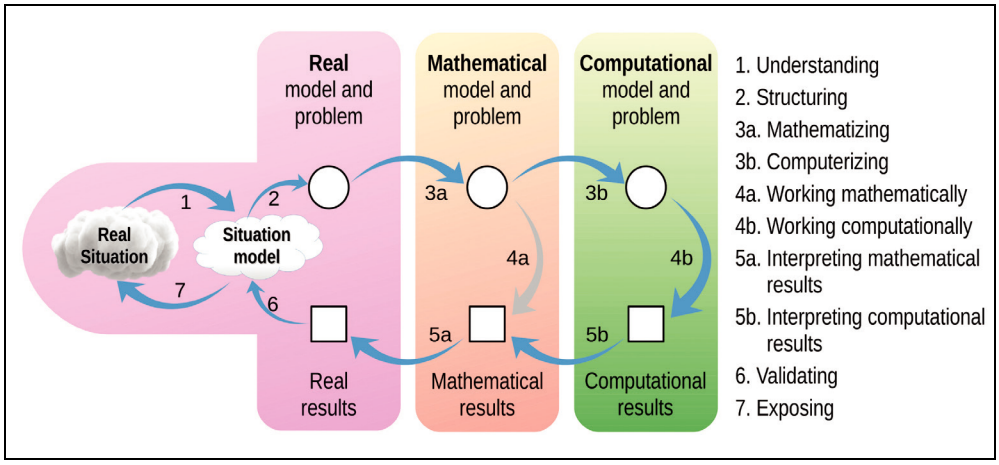

Figure 2. Computational Mathematical Modeling Cycle. Mathematical modeling is an iterative process that involves several steps from identifying and selecting variables to represent a situation to validating the results, repeating the process until the model reaches an acceptable format given time and processing constraints. Figure 2 is adapted from computational mathematical modeling cycle [3].

may revisit their initial assumptions, add or remove parameters, modify both the algorithm and the code and run new tests, repeating the process until they are satisfied with the model.

SETTING THE STAGE

School and teachers - The module was implemented in a public high school located in a fairly prosperous community, with a median annual household income in 2019 above US$170,000 [15]. The school offers a comprehensive four-year program with over 1,500 students enrolled, but only 10% or fewer of students enrolled in a CS course at some point. Asked the reason for the low enrollment rate, one teacher explains: "*It is because of capacity. You don't have the number of teachers, or no sections are approved to run. So we have wait lists for those classes.*" The two participating teachers had 8 and 18 years of experience teaching mathematics at the high school level. One teacher had taught introductory CS courses using Python as a programming language, while the other teacher had no experience in teaching CS or programming. Despite their limited experience in programming, both teachers were enthusiastic about the opportunity to engage their students in using computer programming to solve mathematical modeling problems.

The students - The population in this study is formed by 42 students enrolled in Discrete Mathematics (DM). The DM classes consisted of seniors in the second semester of their final year of high school, enrolled in DM to fulfill their mathematics requirements before graduation. There were 28 females and 14 males, and the vast majority of them intended to pursue careers in the humanities or social sciences (48% and 24%, respectively). Six students said they were interested in pursuing careers in STEM such as marine biology and neurosciences; however, none indicated an interest in majoring in CS or mathematics (Figure 3). While three students mentioned that they had had some contact with programming in the past, 39 students (93%) declared not to have any prior experience with CS or programming.

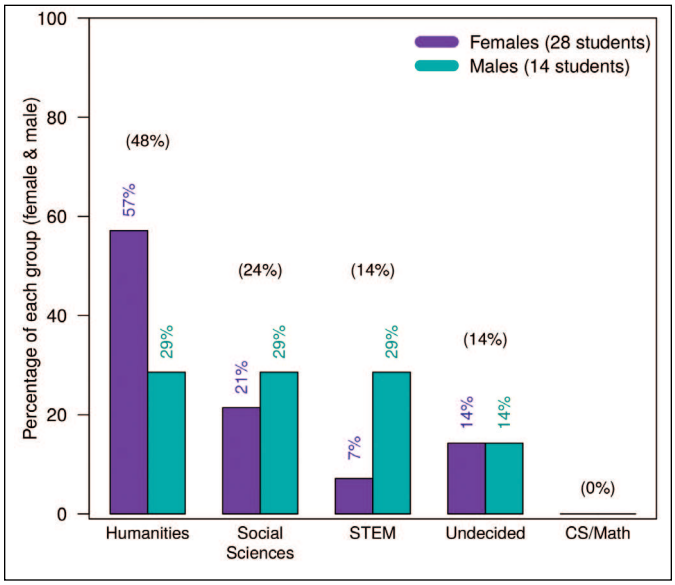

Figure 3. Intended major among students. The female/male ratio of the students was 2:1 in this study, and the vast majority intended to pursue careers in the humanities or social sciences.

METHODS

CURRICULUM AND IMPLEMENTATION

We split the activity into two real-world problems that we named *Meal Plan vs. Pay-As-You-Go* (M.P. vs. PAYG) and the Driving for Gas (DFG), both part of a module we called *Lifehacking*. The module was designed to present open-ended problems relatable to someone who is entering the adult life, like figuring out the costs of eating in college or the real costs of owning and driving a car. The activities were presented to students as online tutorials; therefore, students could work on them from any computer with access to internet. Students worked on this module in a total of 6 hours throughout two and a half weeks.

Either because of socioeconomic status or personal interests, students may think that building a mathematical model to help them with a one-single-time-decision it's just not worth the effort. Paper and pencil and a hand calculator would do. While students may still be willing to do some work for academic purposes (to get a good grade, for example), they would likely be more inclined to embrace a modeling exercise if they could see the model as a tool to help other people while avoiding the tedious work of repeating the same calculations from scratch for each person.

The tutorials were designed using RMarkdown [2] and the tools from the **`learnr`** R package [11]. The tools in the **`learnr`** package allowed us to create code snippets, pre-populate them with templates (complete or incomplete codes), add hints and solutions, and more. Once the student was logged in, any modification in the snippets was saved after they logged out, so they could continue the exercise from the point they left. Figure 4 shows an example of one of the tutorial pages and the kind of platform students were working on, designed to be self-paced with all instructional elements. Students were free to modify the pre-populated code snippets, complete coding exercises, check hints and solutions, and write and run their own code.

MEAL PLAN VS. PAY AS YOU GO

In the first part of the activity, students were invited to wear the shoes of a college advisor who has to help college students decide between purchasing a meal plan (with unlimited access to dining halls with an upfront fixed cost) or paying out of their pocket on a per meal basis. We decided for this approach (suggesting they should work on a solution for multiple people with multiple interests and values) to address the problem we described in session 2.1, that is, they should develop a mathematical model that could (1) maximize the number of college students they could help in a given time and (2) find the best solution for each particular case. Step-by-step, students were guided through the math modeling process, brainstorming, making assumptions about their clients' (the college students) eating habits, budgets, lifestyles, and personal preferences. They gradually refined the model by revisiting their previous premises and adding new parameters such as a variable number of outings per week (dinner with friends, for example), or assuming that their client could have free meals for certain periods, maybe while visiting their family.

Students worked on this activity during five sessions over two weeks and were strongly encouraged to think aloud and discuss in groups. Teachers and researchers followed the group discussion closely, taking notes of behaviors, comments, and drafts that students may have been writing on paper before adding their ideas into code. At the beginning of each new session, the teachers reviewed what they had accomplished, addressed any problems they had encountered, and set the goals for the day. Students worked in small groups through the tutorial at their own pace. All students were able to complete the activity by the fifth session. After this activity, students were presented with a new problem called Driving for Gas (described in the following section) as an assessment of their learning.

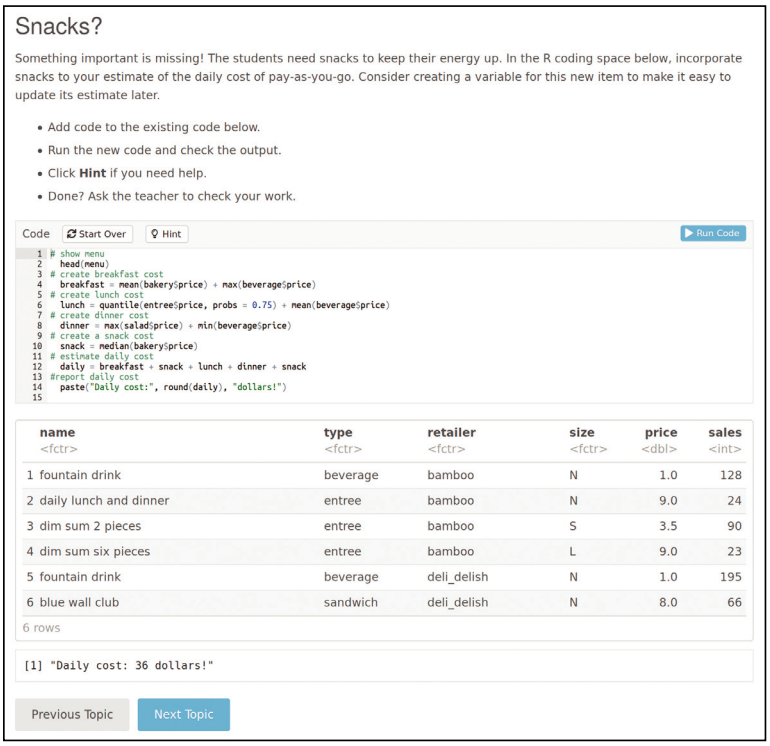

Figure 4. Example of learning activity on CodeR4MATH. The tutorials were designed using RMarkdown and the tools from the **`learnr`** package. And designed to be self-paced, containing all instructional elements.

THE ASSESSMENT: DRIVING FOR GAS

The Driving for Gas activity was adapted from an activity described in the Guidelines for Assessment and Instruction in Mathematical Modeling Education (GAIMME) report [8]. The original problem challenges students to create mathematical models to help drivers decide whether it is worthwhile to drive further to gas stations that sell cheaper gas. To assess the students' modeling skills within a short period, we adapted the problem into a model improvement task. In addition to the original problem statement, students were also presented with a model written as a simple R program. Students were asked to describe how the problem was defined and what variables were considered, comment on the assumptions made, interpret the model's outputs, and improve the model by describing additional factors to incorporate and modifying the R program. Teachers and researchers further elicited students' interest by suggesting that a computer code that would save drivers money and time could become a smartphone or computer app that could become profitable in the future. Students received the suggestion very well and proceeded with their individual work.

Students completed the assessment using RStudio®, a free and open-source integrated development environment for R, which gives them all functionalities they experienced when using snippets, and much more. We asked students to submit their answers individually. The activity requested that students (1) run the template, paying attention to what each part of the model (R code) was doing, (2) interpret the results (a graph), (3) comment and criticize aspects of the model, (4) propose improvements, such as the introduction of new assumptions and variables (or the removal of old ones), and (5) modify the code to implement those improvements.

THE EXIT QUESTIONNAIRE

At the end of the unit, the students completed an exit questionnaire about their impressions, thoughts, and suggestions about the modeling activities and their learning experience. The questionnaire was designed to help researchers understand how the modeling activities had influenced the students' interest in computing and how their backgrounds, such as gender and academic interest, might mediate the impacts.

Students were asked to provide honest feedback for the research team to improve the curriculum. Among other things, students were asked to define mathematical modeling in their own words, to rate their interest in pursuing programming-related courses in the future, and the reasons behind their sentiment toward the activity and computer programming. They were also asked to list the aspects they liked and disliked about the module.

As it is common practice in educational research, classroom observations and discussions with students and teachers were part of the implementation of the CodeR4MATH tutorial. However, the discussion and conclusions presented in this paper are based on the analysis of the modeling assessment and the exit questionnaire.

RESULTS

STUDENTS' MODELING ASSESSMENT

As described in the Methods section, we aimed to assess students' modeling skills using the Driving for Gas activity. The full assessment code, written in R, is shared in Figure 7. First, we asked the students to run the code and summarize the model's outputs. Then, we asked them to suggest modifications and to try and implement their suggestions into the original code. From the initial 42 participants, only 36 students returned the assessment.

Of course, it would be too ambitious to assume that students would become self-sufficient computer programmers after 6 hours of exposure to their first coding experience. The ability to quantify every student's understanding of programming concepts was limited to the students' open answers and comments to the model (the R code). Students were encouraged to write down their thoughts and experiment with the code freely, that is, they were not under any pressure of being graded. On one hand, that allowed the students to brainstorm as they find fit, where many out-of-the-box ideas may emerge. On the other hand, given the limit in time, they didn't feel compelled to either get final quantitative answers or write sharp, focused comments.

Defining which concepts students have (or have not) learned was determined qualitatively. If a student made a reasonable suggestion on how to modify the code in order to improve it (without any judgment on the meaning of *improvement*, which would vary for each student), we added 1 to the *students-who-understood-programming-concepts* bucket. This way our qualitative evaluation of students' responses could be transferred to a quantitative system, more or less binary. Using this method, their comments to the code suggested that half of students (18 out of the 36 respondents) understood components of the model represented in the R program, and that they have achieved some level of understanding of fundamental programming concepts such as the meaning of variables in programming, vectors (as a variable that contains a list of elements), and programming functions (as secondary codes that are given a name and that perform certain tasks when called in the main code). One student, for example, made comments to the code, shown on Figure 5.

```
  (...)
# Calculating labor prices for detour
  travel_price = (.225*distance_add)
  (...)
  cost_gs2_real = cost_gs2 + cost_add + travel_price
  data.frame(distance_add, travel_price, cost_gs2_real, difference)
  (...)
```

Figure 5. Code snippet from a sample of student work. Alterations in the original R code were added by a student with no prior experience in programming.

The code chunk in Figure 5 suggests that the student may have decided to weigh in an additional cost that relates linearly with the distance added to go to another gas station (although they did not comment on the code where the 0.225 factor came from). They then add this factor to calculate the **`cost_gs2_real`** variable and proceed to create a data frame with results and then later (not pictured here) to plot the results. Although

it may seem a very simple modification to the experienced programmer, these students have never had computer programming classes before.

The main output from the original code provided to the students was given by the final graph shown in Figure 6. From the 36 students who delivered the assessment, 15 seemed to have read the model (the R code) as a statement rather than an open problem and may have misinterpreted the output graph, assuming that the second station was at a fixed distance of 30 miles from the first one. "T*he results are strange*," explains one of the students. "*At first, I would have easily made the trip across town to Gas Station 2, however, little did I realize that Gas Station 2 is 30 miles away and ends up being the same price as Gas Station 1.*" Actually, the model describes that the price per gallon is fixed for each gas station, while varying the distance between the two stations (from 1 to 30 miles), to test what would be the real cost of driving to a second station depending on its distance from where the driver (and the first station) is.

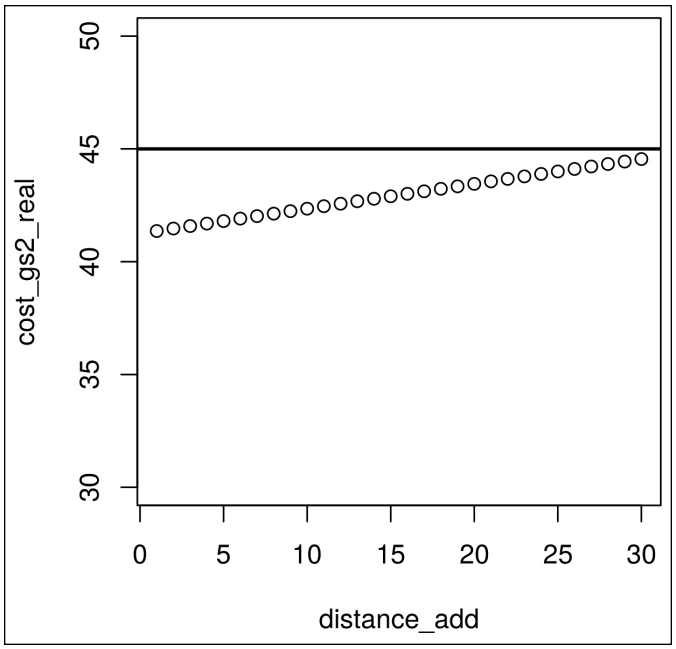

Figure 6. Output from the mathematical model presented in the "Driving for Gas" assessment. Students were asked to run a simplified model whose code was written in R and interpret its output, given by this simple plot.

As we mentioned, students were also asked to describe the modifications they found necessary to make to the model. Students seemed to be forthcoming and were not shy about brainstorming suggestions to make the model more realistic. "*I'd want to factor in the amount of time the person has to go get gas and the amount of money they have to spend*," suggested one student. "*That would make the model more precise and personalized to the needs of the person using it.*" Another student managed to add a few modifications to the code, explaining their thinking, "*I changed the prices of the gas on both gas stations, making it more drastic rather than just 25 cents. I also decreased the distance from gas station 2, so the trip would be more realistic to make. Also, I changed the graph to 10-50 [dollars] for the y-axis, so it's easier to read.*" Their comments suggest their understanding of concepts such as variables and functions in a programming environment, along with a level of appreciation of the ease of testing a model represented in a computer code.

The connection to real-life questions and the ability to look at them through a different perspective but still using the mathematical skills they have already acquired at school appealed to students.

IMPACT OF THE CURRICULUM MODULE ON STUDENTS' INTEREST IN COMPUTER PROGRAMMING

In the exit questionnaire, we also asked students how the modeling activities had influenced their interest in taking computer programming courses in the future. For this question, we provided them a type of Likert scale going from 'I became less interested' to 'I became very interested' and also provided them a field to write down their own thoughts in case they found the options didn't suit their impressions. All 42 students picked one of the first five options. We immediately followed with an open-ended question, asking them to explain what aspects of the modeling activities may have made them more or less interested in taking programming courses in the future.

As described in the Introduction, the participants in this study were seniors, and only two of them had taken any CS course during their high school years. They were enrolled in a standard-level mathematics course to fulfill their math requirement; their mathematics achievement was "*average or below average,*" according to their teachers, and the vast majority of them intended to pursue careers in the humanities or social sciences. All these characteristics projected a development path away from computing education. Yet many of the students came to see the value of computing and found interest in the field that they previously perceived as irrelevant.

A total of 14 students (one third of participants) said they became interested in taking programming classes in the future (sum of *Little more interested* and *Much more interested* categories in Figure 8). Half of the students said that they were not interested in it before the activity and that their inclination did not change. Two students (5%) said that they were already somewhat interested in taking programming classes in the future and that didn't change either. Lastly, five students (12% of the total) said that they became less interested after this activity. The reasons that students indicated to have contributed to their increased or decreased interested in pursuing computer programming are discussed below.

Driving for Gas Assessment

Some of your colleagues have been working on a modeling problem described below:

Gas prices change on a nearly daily basis, and not every gas station offers the same price for a gallon of gas. The gas station selling the cheapest gas may be across town from where you are driving. Is it worth the drive across town for less expensive gas? Create a mathematical model that can be used to help understand under what conditions it is worth the drive.

The R code below is the model your colleagues created. Read the code and run it in your RStudio and answer the questions.

```
# MODEL ----------
# Gas Station 1 (GS1) is right where the driver is.
# Gas Station 2 (GS2) requires a detour.
# The prices of regular gas at GS1 and GS2 (dollar/gallon)
price_gs1 = 3.00
price_gs2 = 2.75

# The amount of gas the driver needs to buy at either station (gallon)
amount = 15

# The costs of buying gas at GS1 or GS 2 (dollar)
cost_gs1 = price_gs1 * amount
cost_gs2 = price_gs2 * amount

# The additional distance for the detour to GS2 (mile)
distance_add = 1:30

# The fuel economy of the driver's car (miles per gallon):
mpg = 25

# The amount of gas used for the detour to GS2 (gallon):
amount_add = distance_add / mpg

# The additional cost for the detour to GS2 (dollar):
cost_add = price_gs2 * amount_add

# The real cost of buying gas at GS2, taking the detour into consideration:
cost_gs2_real = cost_gs2 + cost_add

# Compare the costs of buying gas at GS1 vs. GS2:
difference = cost_gs1 - cost_gs2_real

# REPORT RESULTS ----------
# Display the data generated by the model:
data.frame(distance_add, cost_gs2_real, difference)

# Impact of additional distance on the real cost of buying gas at GS2:
plot(x = distance_add, y = cost_gs2_real, ylim = c(30, 50))

# Visualize the cost of buying gas at GS1 as a horizontal line:
abline(h = cost_gs1)
```

QUESTIONS

1. How did your colleagues define the phrase "worth the drive" in the problem statement?
2. What is the main factor that affects which station will be recommended by the model? Explain why the factor is important to include in the model.
3. Below are some assumptions your colleagues made in order to create the model. Comment on each of them whether they are relevant and reasonable and why you think so.
   a) The driver will buy the same amount of gas at either station.
   b) The driver recently renewed his or her driver's license.
   c) The cost of driver's time is negligible.
4. Based on the model, identify one more assumption you colleagues might have made and comment on its relevance and the reasoning behind it.
5. Run the model in RStudio and summarize the model's results.
6. Do you think the results make sense in reality? Which of the definitions and assumptions that your colleagues made were problematic? Describe how you would address the issue.
7. Modify the R code to make your model generate more realistic recommendations. Feel free to add new variables and expressions and ask the teacher for coding help. Copy and paste your R code, generated data, and plots in the space below.

Figure 7. "Driving for Gas" Assessment. Driving for Gas assessment as presented to the students. They were required to comment on the R code, suggest modifications and refinements, and finally implement them to the code.

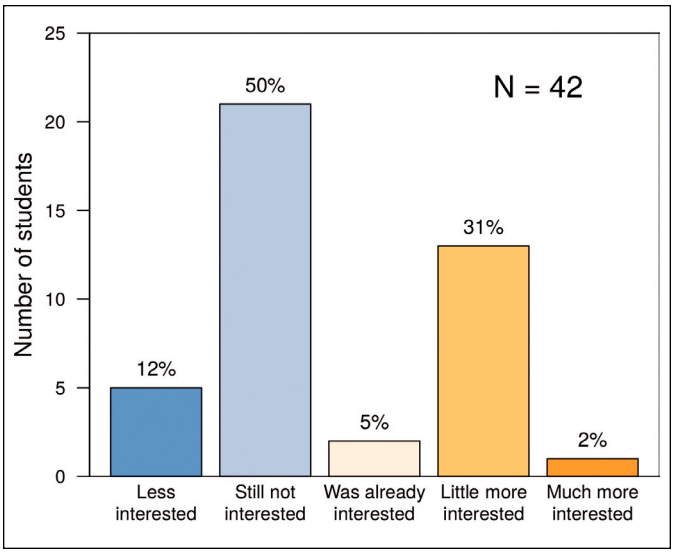

Figure 8. Impact of the mathematical modeling activity of students' interest in computer programming. Many students came to see the value of computing and found interest in a field that they previously perceived as irrelevant to them. One-third of the students said that they became interested in taking programming classes in the future.

THE REASONS BEHIND INCREASED OR MAINTAINED INTEREST IN PROGRAMMING

Students were also asked to explain what about the learning experience made them more or less interested in taking computer programming courses in the future. We categorized their open-ended responses into broad themes. The three main themes for which students became interested in computer programming were:

- **Coding is powerful**
- **Solving real-world problems with math modeling and coding**
- **New experience**

Positive reason 1: Coding is powerful

Eight students said that coding is a powerful tool that makes data analysis much easier and faster. The majority of students who saw coding as a powerful tool was disproportionately male (75% within this subgroup). They also mostly intended to major in fields that require quantitative skills (e.g., STEM and social sciences). "*I just found it really fascinating how coding can make life that much easier*," said a student interested in majoring in business. "When I learn more and more about it, I get more into it. Something I don't experience every day, it's nice and in a way fun." Another student, who intended to major in marketing, said that they appreciated "*how you can use coding to solve big mathematical problems*." Another student, who claimed to be interested in STEM, said that they became more interested in programming after the activity because "*coding can make keeping track of data much easier*."

Positive reason 2: Coding can be used to solve real-world problems

The connection to real-life questions and the ability to look at them through a different perspective but still using the mathematical skills they have already acquired at school appealed to students. Several previous studies (e.g., [9] and references therein) have already shown that solving real-life problems often increase students' performance in CS courses. "*I thought it was very interesting*," a student said. "*It opened up my mind to other ways in which math can be used in the real world*."

Students also appreciated the process of mathematical modeling, which relates directly to the exercise of CT skills. "The coding exercises required that I defined all variables and then integrate them into some sort of data structure, which is a valuable process to utilize even without coding," said one of the students. Also, the teachers have appreciated that students could experience the thinking process of solving an open-ended problem. "*I think getting them to think of a big picture for me was probably the best piece,*" one of the teachers said. "*It was just getting them to think bigger than themselves. How do I take this problem, and how do I project it, and what are the issues I need to think about?*"

"*I think that connecting, getting them to brainstorm, getting them to be more aware, that was my favorite part*," another teacher said. "*Having them [students] think that there are things out there that you can add value [to], that people's days can be better because of your model*."

Positive reason 3: It was a new experience

Finally, the exposure to something that they had never done before also seems to have captivated students' interest. "*It was a new and unique experience that showed a new way of thinking about modeling,*" said a student, "*I knew nothing about computer programming, so it gave me a look into what it can be*."

Students seem to have found it compelling to have been exposed to a new use of a technology with which they are already familiar (their computers) to do something different and more applied. "*I don't take any classes that require [me] to do anything on my computer besides type essays or Google things*," another student said, "*so using my technology in a different way was cool for me*."

REASONS BEHIND SOME STUDENTS' DISINTEREST IN PROGRAMMING

Not all students felt compelled to explore more programming-related courses though. Twenty-one participants (50%) said that they were not interested in computer programming before the activity and that they were still not after the exercise. Five students stated that their interest in programming had actually declined. The reasons these students pointed out for not becoming interested (or becoming less interested) in programming vary, but the vast majority were related to their self-concept in relation to CS and mathematics itself. In fact, the responses from over 80% of students in this group (still not or less interested in programming) fall under this self-concept category, and the disproportionate majority of these are females (a 6:1 female-male ratio, compared to 2:1 for the whole population that responded to the questionnaire).

Negative reason 1: Self-concept

Some students seem to believe they are simply not good at mathematics or computer sciences. "*I've never been a heavy math person*," said a student interested in the humanities, "*and although computer coding is intriguing and seems complex and fascinating, I have no real interest to take another class*." Still, they recognize that being exposed to the experience was helpful to them, adding that "*I liked that it was unlike anything we have done all year, it was new and yes a little confusing but also interesting*." Another student says that "*Coding is just something I'm generally not interested in, but I think it was good that I was introduced to it, just in case I may need to use it*."

We can say that even if for part of the students, the interest in programming has not been triggered, they became aware of the value of learning about the process of math modeling and coding.

Negative reason 2: More scaffolding, please!

The activities were designed to pose semi-open questions to the students, in the sense that students were guided with instructions and examples, and their teacher-led discussions in the classroom. Because the tutorials were self-paced, beginners could take their time to walk through the activities, while more able students had the opportunity to go quickly through the basics and dedicate more time to refining their models. A few students, however, indicated that they would prefer more scaffolding and more direction from their teachers, showing some level of discomfort with the curriculum design. "*Even when I asked questions, it was still extremely unclear on what to write within the code and what I was supposed to be analyzing*," said a student who was interested in the performing arts. When asked what they did not like about the activity (another question in the exit questionnaire), they pointed out that "*when working with code or when creating code, you aren't necessarily looking for one specific answer. That really tripped me up and was something I didn't understand until the last day of coding*."

Negative reason 3: Coding is boring

Two students who intended to major in the humanities remarked that, for them, programming was boring and not useful.

"*I found it boring, and there was no creative part of it*," said one of them. They continued by saying that "*I didn't find mathematical modeling to be that much of use*." A second student said, "*it was very confusing, and I did not think this had anything to do with this math class*."

Except for the students who felt they would prefer another delivery format, with more scaffolding (likely from their teachers), all other reasons given by students for not liking computer programming seem to be primarily due to preconceived ideas about the subject or about themselves. These students apparently see programming, and even mathematics, as subjects reserved to the tech-savvy or to the *math person*, believing that some people are naturally good at mathematics and others are simply bad at it, and put themselves in the latter group.

FUTURE TOPICS OF INTEREST

Finally, we asked students about their topics of interest, that is, subjects they would like to learn more about and that they would like to deal with in an activity like this. Forty-five percent of participants (19 students) said they would like to explore issues related to daily life problems, such as finances, student life, or personal time management.

"*Budgeting in everyday life, taxes, and more*," suggested one of the students. "*Topics that adults deal with in everyday life, but that aren't necessarily explained and/or taught to students within the school curriculum*." This student was interested in pursuing a career in the performing arts. These participants identified possible topics of interest as being related to a person's daily life, in the lines of the *Lifehacking* curriculum module. Another student who declared their intention to follow a career in psychology said, "*Since I am going to college, another topic on it and saving money would be engaging*," and said that they became more interested in exploring programming-related courses in the future.

DISCUSSION

There is no substitute for a motivating environment where students feel empowered to do great things with the resources that they already have in hand. The use of code snippets to reduce syntax-related stress, along with engaging and relatable real-world problems seem to have helped many students feel comfortable and curious about the possibilities of math modeling and programming, while exercising computational thinking skills. While the activity was not able to embed appreciation for modeling and/or programming to all participants, we were pleasantly surprised to find that one-third of them indicated they actually became more interested in pursuing programming-related courses in the future after this experience. This is a positive and encouraging outcome, considering that most of these students had not taken computer science courses during their high school years and the vast majority of them had not seen how computing was relevant to their everyday life and intended majors.

This study was not intended to investigate how non-CS teachers could be prepared to teach CS content in their courses such as mathematics course. However, in close collaboration with the teachers, we observed strong potential for non-CS teachers to develop competencies to integrate CS content in their classrooms using the designed tutorial, without typical, extensive professional development experience. While one of the teachers had no prior experience in teaching CS, they both showed the same high autonomy in using the tutorial and guiding their students to complete the activity. Students' performance in the assessment and their responses to the exit questionnaire were comparable for both classrooms. Despite the small sample size in this study, these results strongly suggest that the classroom-readiness of the tutorial materials is critical to ensure not only that students will have a worthy learning experience, but also to boost teachers' confidence in repeating the implementation or even adapting the curriculum with their own inputs or style, whether or not they have training in teaching CS.

The overwhelming majority of students who said they were not interested in programming (or became less interested) were females (~78%). Although we do not focus on the gender gap, it did not pass unnoticed. However, it would be incautious to affirm that gender may have played a major role in this result, because of confounding factors such as the fact that most of them were interested in careers in the humanities. They formed the vast majority of students whose reasons behind their lack of interest in programming are extended to the sciences in general. Several responses included phrases like "*I am not a math person*," or "*It doesn't play to my strengths*," and "*I don't think it is for me*." It would be too ambitious to expect that educators could deconstruct the *I-am-not-a-math-person* mindset created by youngsters who reached their high school years. However, we hope that the results from this experience can provide some insights on how to exercise computational thinking in mathematics classrooms to help students abandon the erroneous preconceived idea that exact sciences (and programming in particular) are out of their reach. The mathematical modeling process is naturally aligned with computational thinking, and math modeling activities regularly naturally motivate the use of computational tools. Our results reinforce that, although it may seem counter-intuitive, presenting students with CT approaches to solution building (e.g., visualizing a set of data, finding a multitude of solutions in moments for comparison and/or analysis) in their regular math classes have great potential to lead to better learning outcomes.

SUMMARY

Computational thinking is a collection of thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be carried out by an information-processing agent; a computer program executed by a computer, for example. Of course, computational thinking is a highly desirable skill set, and an excellent way to exercise this way-of-thinking is through math modeling and computer programming. However, students are leaving high school without having much experience—if any—with programming or, more broadly, with computer sciences. That

may happen for several reasons. In one scenario, students simply don't have classes being offered at their school. In another situation, a school may offer computer science classes, but students are either too busy with their regular classes or think that because of their career preferences for the future, programming (or mathematics itself) is "not for them." To overcome these barriers, the CodeR4MATH project proposes to integrate computational thinking into regular, mandatory mathematics classes. We propose to give students the opportunity to create, visualize, and analyze mathematical models using the programming language R as the modeling environment. At the same time, activities like the one presented in this study, provide teachers with several instructional tools that they can use in the classroom to promote integration between CT and CS into their regular math classes, without having to open up room to a foreign subject.

Here we described the results from the implementation of our pilot module called CodeR4MATH on a population of 42 senior high school students enrolled in discrete mathematics. The module is composed of several self-paced tutorials that challenge students to model solutions for practical problems like the cost of eating in college, the real costs of owning a car, deciding between careers, etc. In other words, the module was designed to bring open-ended problems that are relatable to someone entering adult life. All but three students had no prior experience with programming, and the whole group had never enrolled in computer science classes before.

Our results are undoubtedly encouraging. After a total of six hours of interaction with the tutorials, including an assessment, a third of the students who participated in the implementation declared that they became interested in taking programming classes in the future (2 students, or 5%, said they were already interested in exploring this possibility in the future).

Students cited a few reasons for becoming interested in programming that we put into three major groups: they felt that coding is a powerful tool, that it can be used to work on and solve real-world problems, and some students simply became interested because the whole experience was out-of-the-ordinary use of technology, which was something new to them.

We also observed that for the rest of the students, that is, those who did not care about programming and that said their interest did not change, the primary reason we identified was self-concept. Twenty-one out of the 26 students who said that they were still not (or less) interested in programming, justified their sentiment because they believe that some people are naturally good at mathematics and others are simply not, identifying themselves as not being a *math person*. Although it is common sense that there is no such thing as a math person, it is unfortunate that these youngsters may leave high school (whether or not they decide to go to college after that) without developing true appreciation for exact sciences. It may be difficult for a high school teacher to change this mindset on students who reached high school and are preparing to ingress in the adult life. However, we hope that the results from this classroom experience can provide some insights to teachers so that they can help their students to abandon the erroneous preconceived idea that exact sciences (and programming in particular) are out of their reach. ❖

Acknowledgments
This research was supported by the National Science Foundation under grant number 1742083. We are grateful to the numerous discussions with mathematics teachers during curriculum implementations, workshops, conferences, and alike. Their support, knowledge, and passion make our work possible.

References

1. Akpinar, Y. and Aslan, Ü. Supporting Children's Learning of Probability Through Video Game Programming. *Journal of Educational Computing Research*, 53, 2 (2015).
2. Allaire, J.J., Xie, Y., McPherson, J. Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., Chang, W., and Iannone, R. (2019). rmarkdown: Dynamic Documents for R. R package version 1.14; https://rmarkdown.rstudio.com. Accessed 2019 July 26.
3. Blum, W. and Leiβ, D. (2007). How do students and teachers deal with modelling problems? In C. Haines, P. Galbraith, W. Blum, & S. Khan (Eds.), *Mathematical modelling (ICTMA 12): Education, Engineering and Economics*. (Chichester, UK: Horwood Publishing, 2007), 222-231.
4. Bruner, J. S. From communication to language - a psychological perspective. *Cognition*, 3, 3 (1974), 255-287.
5. Bureau of Labor Statistics Employment Projections data for 2018-2028; https://www.bls.gov/ooh/computer-and-information-technology/computer-and-information-research-scientists.htm. Accessed 2019 October 30.
6. Google Inc. & Gallup Inc. Trends in the State of Computer Science in U.S. K-12 Schools; https://news.gallup.com/reports/196379/trends-state-computer-science-schools.aspx. Accessed 2019 July 26.
7. Gilbert, J.K. On the Nature of "Context" in *Chemical Education, International Journal of Science Education*, 28, 9 (2006), 957-976.
8. Guidelines for Assessment & Instruction in Mathematical Modeling Education (2016); https://www.siam.org/Publications/Reports. Accessed 2019 August 6.
9. McMaster, K., Anderson, N., and Rague, B. Discrete math with programming: better together. *ACM SIGCSE Bulletin*, 39, 1 (2007),100-104
10. R Core Team. R: A language and environment for statistical computing. *R Foundation for Statistical Computing*, Vienna, Austria; https://www.R-project.org/. Accessed 2019 July 26.
11. Rose D.E. Context-Based Learning. In: Seel N.M. (eds) *Encyclopedia of the Sciences of Learning*. (Springer, Boston, MA, 2012).
12. Schloerke, B., Allaire, J.J., and Borges B. **`learnr`**: Interactive Tutorials for R. R package version 0.9.2.1; https://CRAN.R-project.org/package=learnr. Accessed 2019 July 26.
13. State of Computer Science Education (2018); https://code.org/files/2018_state_of_cs.pdf. Accessed 2019 July 26.
14. Tawfik, A.A., Law, V., Ge, X., Xing, W., and Kim, K. The effect of sustained vs. faded scaffolding on students' argumentation in ill-structured problem solving. *Computers in Human Behavior*, 87 (2018), 436–449.
15. United States Census Bureau (2018); https://www.census.gov. Accessed 2019 August 9.
16. Wing, J. Computational Thinking. *Communications of the ACM* 49, 3 (2006), 33-35.

Kenia Wiedemann
Research Associate
The Concord Consortium
25 Love Lane, Concord, MA USA
kwiedemann@concord.org

Jie Chao
Research Scientist
The Concord Consortium
25 Love Lane, Concord, MA USA
jchao@concord.org

Benjamin Galluzzo
Associate Professor
Institute for STEM Education
Clarkson University
8 Clarkson Avenue, Potsdam, NY USA
bgalluzz@clarkson.edu

Eric Simoneau
CEO at 33 Sigma Labs
22 College Avenue, Arlington, MA USA
Mathematics Teacher at Boston Latin School
78 Avenue Louis Pasteur, Boston, MA USA
mrsimoneau@gmail.com

DOI: 10.1145/3380956