Implementing Unplugged CS and Use-Modify-Create to Develop Student Computational Thinking Skills: – A Nationwide Implementation in Colombia

Camilo Vieira[1*], Ricardo L. Gómez[2], Margarita Gómez[3], Michael Canu[4] and Mauricio Duque[3]

[1]Universidad del Norte, Colombia // [2]Universidad de Antioquia, Colombia // [3]ACCEFYN, Colombia // [4]Universidad El Bosque, Colombia // cvieira@uninorte.edu.co // rleon.gomez@udea.edu.co // mgomez@stem-academia.net // mcanu@unbosque.edu.co // ismaduque@stem-academia.net

[*]Corresponding author

(Submitted February 17, 2022; Revised July 18, 2022; Accepted November 20, 2022)

ABSTRACT: This paper describes the implementation and student learning outcomes of a nationwide professional development program for lower secondary and upper secondary school teachers to integrate computational thinking into the K-12 curriculum. Computational thinking comprises important concepts and skills that all students should develop to take an active role in a global society. However, teaching computational thinking is challenging. There are few teachers with the knowledge and skills to integrate computation into their courses. In this program, the participating teachers implemented a set of lesson plans that included both unplugged activities to scaffold student learning, and ‘plugged’ activities following a use-modify-create learning progression with the Micro:bit device to practice these skills. The study used a quasi-experimental design to compare students’ level of computational thinking between the program participants and a control group. The results suggest a positive effect of the learning activities on student computational thinking knowledge and skills as compared to the control group. This result persists after controlling for school context and student gender. This study provides an explicit approach to implementing these activities in the context of a developing country and assesses their effectiveness in a large-scale study.

Keywords: Computational thinking, Micro:bit, Use-modify-create, Unplugged, Teacher professional development

1. Introduction

Computational thinking (CT) is an important set of skills and practices that enable professionals from all disciplines to solve complex problems. Wing (2011) defined it as “the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent” (p. 1). Different definitions for computational thinking often involve concepts and practices such as abstraction, decomposition, automation, algorithmic thinking, and modeling and simulation (e.g., Grover, 2017; Katai, 2020; Noh & Lee, 2020). Such practices allow professionals to process large datasets, automate repetitive tasks, and understand and predict complex phenomena using, modifying, and creating computational models and simulations (Weintrop et al., 2016). Hence, computation has been denominated the third pillar of scientific discovery (Wing & Stanzione, 2016), together with theoretical and experimental approaches to inquiry.

The affordances of computing for professionals in all disciplines have generated a call from academics, governments, and international agencies to integrate computational thinking across all educational levels (Angeli et al., 2016; Barr & Stephenson, 2011; Katai, 2020; Lee et al., 2011; Royal Society, 2012). Some authors even suggest that developing student computational thinking is as important as learning math or reading and writing (Grover & Pea, 2013; Sanford & Naidu, 2016). However, teaching computational thinking may be more effective and inclusive when integrated into disciplinary contexts and real-world problems (Jocius et al., 2021; Mouza et al., 2020; Katai, 2020; Weintrop et al., 2016). Nevertheless, some degree of explicit direct instruction of basic concepts is required to avoid distraction and cognitive overload in students trying to make sense of a contextual project (Tricot, 2017).

Several countries, including the United States (Mouza et al., 2020), the United Kingdom (Curzon et al., 2014), and Australia (Yadav et al., 2017), have started to integrate computational thinking into their K-12 curricula. In many of these places, however, there is no formal training for teachers to incorporate computational thinking, which offers challenges for teaching it at the schools, and may stress teachers (Angeli et al., 2016; Caeli & Bundsgaard, 2020). Furthermore, most of what we know about how students learn computing is from developed countries and privileged students. More research is required to understand how students from different contexts (e.g., culture, language, technology access) learn about computational thinking and their challenges in this process.

This paper reports on the student outcomes after implementing a professional development program for lower secondary and upper secondary school teachers aimed at integrating computational thinking concepts and practices into the public school system in Colombia: “Programación para Niños y Niñas” (in English, Coding for Kids - CFK). Colombia, like many developing countries, faces four interrelated challenges. First, the education system is not producing enough people with CT skills to fill both current and projected industry demand. Second, women are significantly underrepresented in the STEM education pipeline and workforce. Third, many teachers have no preparation to integrate CT into their courses. Fourth, there is limited research on STEM teachers’ pedagogical knowledge (PK) of CT and its effect on student learning.

Colombia is a country with significant achievement gaps between men and women and between urban and rural students. Colombia is the country with the most critical situation among all the countries belonging to the OECD: although Colombia follows the world trend of a better performance of women compared to men in Language skills, it has the smallest gap in favor of women. While in science and mathematics, the country presents the largest gap against women (OECD, 2020). The results of the 2018 PISA test also showed a systematic and large gap of almost 40 points between students in urban and rural areas, a value that compares with 24 points at the average level of OECD countries (ICFES, 2020). Bridging this divide between CT skills and those that students currently develop in Colombian schools requires innovative approaches to K-12 education (Barr et al., 2011; Mohaghegh & McCauley, 2016).

This paper aims to address these issues and assess the effects of a national-level professional development program on student learning. The research team designed a set of lesson plans that integrate computational thinking into their learning environments. Computer programming and CT skills represent a form of complex learning, where students need to consider many interacting elements at once (Mselle & Twaakyondo, 2012). Reducing extraneous cognitive loads and facilitating the development of schemata may contribute to effectively supporting student learning (Sweller et al., 2019). The lesson plans included a use-modify-create progression to scaffold student learning and reduce extraneous cognitive loads. While use-modify-create has been suggested as a promising approach to introduce CT skills (Lee et al., 2011), empirical evidence of how it works has just started to emerge (e.g., Franklin et al., 2020; Lytle et al., 2019), and little is known about how to operationalize its implementation. In this paper, we describe the design of Coding for Kids (CFK), its theoretical underpinnings, and its effects on students’ CT skills. This study contributes to the body of knowledge of computational thinking education by providing and assessing an approach to integrate both unplugged learning activities and the use-modify-create learning progression in a developing country like Colombia. While we describe the teacher professional development program, our ultimate goal was to understand how it impacted student learning. To our knowledge, there is no such large-scale implementation of a professional development program that assesses student learning in a similar context. In this context, both teachers and students have limited knowledge about computing; the main language is Spanish (very few understand English), and the school infrastructure is limited. The guiding research question for this study is:

What is the effect of the Coding for Kids program on student computational thinking knowledge and skills?

2. Theoretical framework

2.1. Constructivism and cognitive load theory

The CFK program is based on the theoretical frameworks of both constructivism and cognitive load theory (CLT). Both of these frameworks informed the design of the lesson plans and can help explain potential differences in student performance on CT skills. From a Constructivist perspective, students build their knowledge starting from their prior knowledge while engaging in reflective practices (Ben-Ari, 1998; Jonassen et al., 1999). These student-centered learning environments promote changes in the existing cognitive structures and internal representations. Making connections to prior knowledge – including a mental model of what a computer is – and to relevant contexts for the student is key from this perspective (Ben-Ari, 1998).

Likewise, the Cognitive Load Theory (CLT) suggests a cognitive architecture that we use to process information and learn new concepts and skills (Sweller et al., 2019). The cognitive architecture includes a working memory (STM) and a long-term memory (LTM). The STM is limited in time and space, while the LTM is vast. There are different types of cognitive loads in the STM. The intrinsic load is given by the complexity of the concepts or skills to be learned, and can hardly be modified (Sweller et al., 2011). The extraneous load is not required for learning, and is often given by poor instructional designs or poor learning materials (e.g., trying to make sense out of the relationship between a poor figure and a text). The germane load, also called germane resources, is the positive load for learning. The germane resources help us make sense of the intrinsic load, by making connections to existing schemata (i.e., how knowledge is stored in the LTM). When we need to learn many interacting elements at once, we can experience a cognitive overload in our STM. We can only remember what we have in the STM if we make significant connections to the schemata stored in the LTM.

The CLT suggests several effects that may derive into effective pedagogical strategies to manage the cognitive loads in the STM. For instance, the worked-example effect describes how novice learners may reduce their cognitive loads by engaging in an active exploration of worked-examples instead of engaging in problem-solving from scratch. A worked-example is an expert's solution to a problem, which includes a problem statement, a step-by-step solution, and auxiliary representations of the problem and the solution (Atkinson et al., 2000). The worked-example effect suggests that by first exploring an expert's solution to a problem, novice learners start developing the required schemata that enable them to solve problems independently. At some point, the learning environment may include a fading approach by providing only partially solved worked-examples: the completion effect.

2.2. Implications for the lesson plans

We used two strategies in the design of the lesson plans to manage student cognitive loads. Start with unplugged learning activities to present basic concepts. Continue with computing activities using the Makecode editor and the Micro:bit device following the "use-modify-create" progression. The Micro:bit is an open-source programmable microcomputer to teach computer science concepts (Sentance et al., 2017). Physical computing devices such as the Micro:bit device have been explored to introduce computing concepts, and suggest positive outcomes on reducing student cognitive load while promoting creativity and motivation (Sentance et al., 2017). While these devices may be useful for computational thinking education, they require a pedagogical integration to support student learning. We designed seven lesson plans that teachers implemented in two separate 50-minute sessions. The first session is dedicated to an unplugged activity, while the second session is focused on the use-modify-create learning activity. Two additional lesson plans focused on assessment activities for teachers to monitor student learning.

This study explores the effect of integrating unplugged learning activities in computational thinking together with the "Use-Modify-Create" strategy on students' skills in Colombia. Hence, this study will contribute to understanding how these two approaches may support the development of CT skills, while providing a concrete approach to operationalize these approaches as a set of activities and scale it to a national level. This is particularly important in the context of a developing country such as Colombia, since most of the existing literature focuses on privileged students from developed countries.

2.2.1. Unplugged CS

Unplugged computer science is an approach to designing learning activities without needing a computing device. Computational thinking includes several skills and reasoning processes, many of which can be approached independently of the artifact: abstraction and generalization of patterns, information processing, symbolic representations and symbolic representation systems, algorithmic thinking, problem decomposition, iterative, recursive and parallel thinking, and conditional logic. Students can develop these skills without the need for a computing device. Instead, students may do "by hand" what a computer program would do in an automated way (Aranda & Ferguson, 2018; Faber et al., 2017). This approach ensures that the focus is on understanding and skill development and not on the artifact at an initial stage of the learning process (Bell et al., 2009). Starting with unplugged activities to learn these skills beyond a specific tool or programming language may help students to develop their problem-solving skills, and reflect on their own thinking (Caeli & Yadav, 2020).

Unplugged learning activities have been suggested as being effective both inside and outside the classroom at different levels to develop a fundamental understanding of concepts about algorithm design, and to change conceptions about the nature of computer science (Caeli & Yadav, 2020). This approach may reduce extraneous cognitive loads from elements that are inherent to some programming languages, like syntax or memory management. These activities also offer an efficient way to attract students to computer science concepts without the need for much time and resources (Bell et al., 2009, Brackmann et al., 2017). Despite the widespread use of unplugged activities, there is little empirical evidence in classroom settings (Bell & Vahrenhold, 2018).

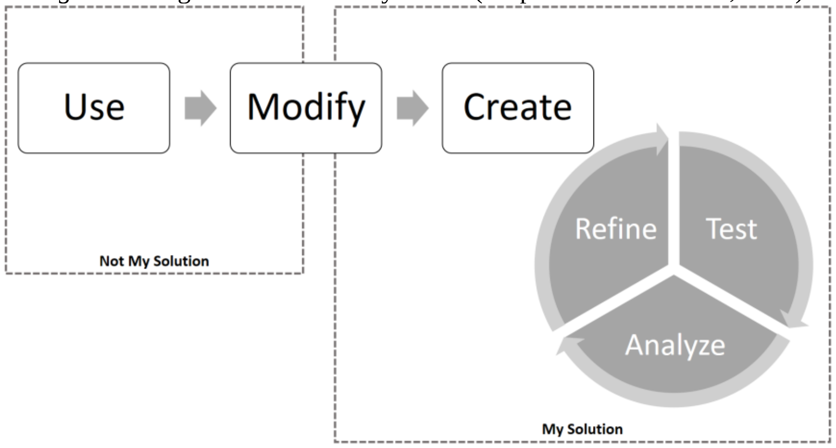
The development of most concepts and practices of computational thinking can be worked unplugged so that students can generalize these skills to other contexts that are not explicitly computer programming. However, researchers recommend using CS unplugged activities in conjunction with programming challenges (Bell & Vahrenhold, 2018). This should not be a discussion of whether CS unplugged is more powerful than using plugged programming lessons alone. CS unplugged activities may help students lower the cognitive loads associated with the device and programming language before actually applying this knowledge using a computer device.

Furthermore, integrating computational thinking at the k-12 level, especially in developing countries like Colombia, often brings challenges related to the accessibility to information and communication technologies for schools and students. This is especially the case for rural populations. At the beginning of this project, only 70% of the participating rural schools had working computing devices for their students (compared to around 90% for urban schools), and less than 50% of all participating schools had internet access. The program designed a series of unplugged activities to address this challenge and scaffold student learning.

2.2.2. Use-Modify-Create

The instructional principles of the worked-examples (Atkinson et al., 2000), explained by the CLT, support the Use-Modify-Create progression to scaffold student learning of complex concepts and skills. Computer programming is a complex skill to learn (Mselle & Twaakyondo, 2012), which may lead to cognitive overload. A novice learner needs to learn about algorithm design, the problem they are trying to solve, the programming language syntax and semantics, Boolean logic, and even how the computer works, all at the same time. The worked-example effect and the completion effect can be mapped to this progression as follows: the instructor provides a complete worked-example for students to explore (Use), students engage in the modification or completion of a worked-example (Modify), and then design and implement (Create) their own solutions iteratively (Figure 1).

Figure 1. Progression use-modify-create (adapted from Lee et al., 2011)



The progression Use-Modify-Create may help reduce such cognitive overload experienced by novice programmers. A recent study compared the use-modify-create progression to a sequence of activities where students always needed to create their own code (Lytle et al., 2019). In this study, the teachers in the create-only condition faced challenges supporting their students, and suggested that students needed additional scaffolding to explore and visualize a solution before they engaged in creating it. Conversely, teachers in the use-modify-create condition suggested that students were able to develop an understanding of how the program worked, and they connected everything once they completed the Create activity. Franklin and colleagues (2020) also found that this progression successfully scaffolds student learning, while promoting student agency. Finally, our own work has also shown that engaging students in the active exploration of examples may support the development of basic schemata for novice learners (Vieira et al., 2019).

Other approaches to support student learning based on the Cognitive Load Theory include sub-goal labeling (Morrison et al., 2015) and self-explaining (Vieira et al., 2017). However, the Use-Modify-Create provides a roadmap for students to increasingly develop their skills and agency (Franklin et al., 2020), moving from studying complete worked-examples, to faded solutions, to creating and refining their own solutions iteratively. The implications of this framework for this study are reflected in the learning design presented in the methods section.

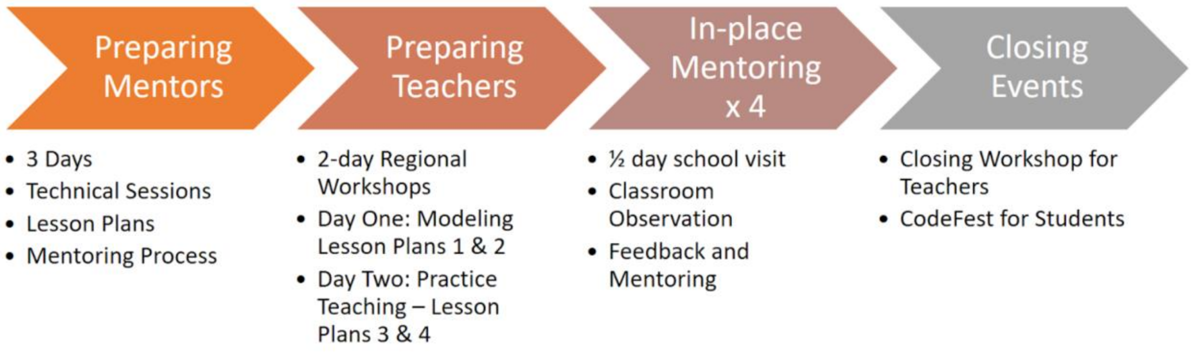
3. Methods

3.1. Program design

Despite the relevance of integrating computational thinking into the K-12 curricula, and the governmental efforts to make it happen, there is no agreement about what practices and skills computational thinking really entails (Curzon et al., 2019; Denning, 2017; Mouza et al., 2020). This makes it difficult to establish a universal set of learning outcomes to integrate into the school curricula (Mueller et al., 2017). For this project, the research team operationally defined computational thinking as a multidimensional set of complex skills, including computational problem-solving, computational modeling, abstraction, decomposition, algorithm design, programming and debugging, and solution validation. This definition integrates common concepts and skills that often appear in several authors' definitions (e.g., Denning, 2017; Grover, 2017; Katai, 2020; Weintrop et al., 2016).

Using this operational definition, the research team designed nine lesson plans to be implemented by participating teachers. While this paper focuses on student learning, the professional development program became our delivery mechanism for the learning activities. Figure 2 summarizes the implementation process of the learning materials. We first prepared a group of 12 mentors to lead the teacher training program. These 12 mentors implemented a two-day regional workshop for the teachers. The mentors modeled lesson plans #1 and #2 on the first day of the workshop, where teachers played the role of students. The participating teachers then prepared and taught lesson plans #3 and #4 for practice teaching on the second day of the workshop. The mentors also visited the teachers in their schools five times in a period of six months. Four of these monthly visits aimed at preparing the upcoming lesson plans, characterizing the participants' teaching practices and providing feedback and mentoring over the lesson plans. The goal of the fifth visit was to collect data on student performance.

Figure 2. Professional development program Coding for Kids



Each lesson plan comprises two 50-minute sessions, and follows the structure presented in Figure 3. The first session focused on unplugged activities, while the second session included plugged activities with the Micro:bit using the use-modify-create progression. The teachers start by discussing with students the learning outcomes for the activity and the required prior knowledge to complete it. This part allows students to manage their expectations, connect to their prior knowledge and, with a reflection at the end, promote metacognitive processes (Robins, 2019). Students work on the Micro:bit on the second session, following a use-modify-create progression. Students first explore an example, predicting the outcome of a program or explaining it to a partner (i.e., Use). Students then make specific changes to the sample program, to engage in a scaffolded problem-solving activity (i.e., Modify). Finally, students complete a challenge from scratch, engaging in problem-solving with the Micro:bit (i.e., Create), using what they learned from the example.

Figure 4 shows an example of this progression in lesson plan #3. The goal of this activity is to simulate an autonomous vehicle for scientific exploration inside a cavern. The proximity sensors are simulated using two buttons, while the Micro:bit display will show where the vehicle would go. Students first need to predict what the sample program is doing and run the program to validate the prediction (i.e., Use). This program identifies when a button is pressed and shows an arrow pointing toward the East. The next step is to Modify this program so that it shows all directions on the display depending on what buttons are pressed. Finally, students should Create a program using the compass sensor to guide the vehicle in one direction (e.g., North) following a black line. Both the Modify and the Create steps in the progression engage the student in an iterative process of implementing, testing, reflecting, and refining, often discussing them with a partner. The learning goals of this lesson plan included: Using Boolean input variables; Communicating instructions using the LED screen; Interpreting a sequence of instructions and a flow diagram to solve a problem such as a maze; Using logical operations; Using loops that are repeated until the task is finished. Students started with the unplugged activity, following and adjusting a flow diagram to move a toy around a sample maze to solve this challenge. These flow diagrams included loops and logical operations that students needed to program later, using the Micro:bit input variables (e.g., buttons) and LED display.

Figure 3. Structure of the lesson plans for Coding for Kids

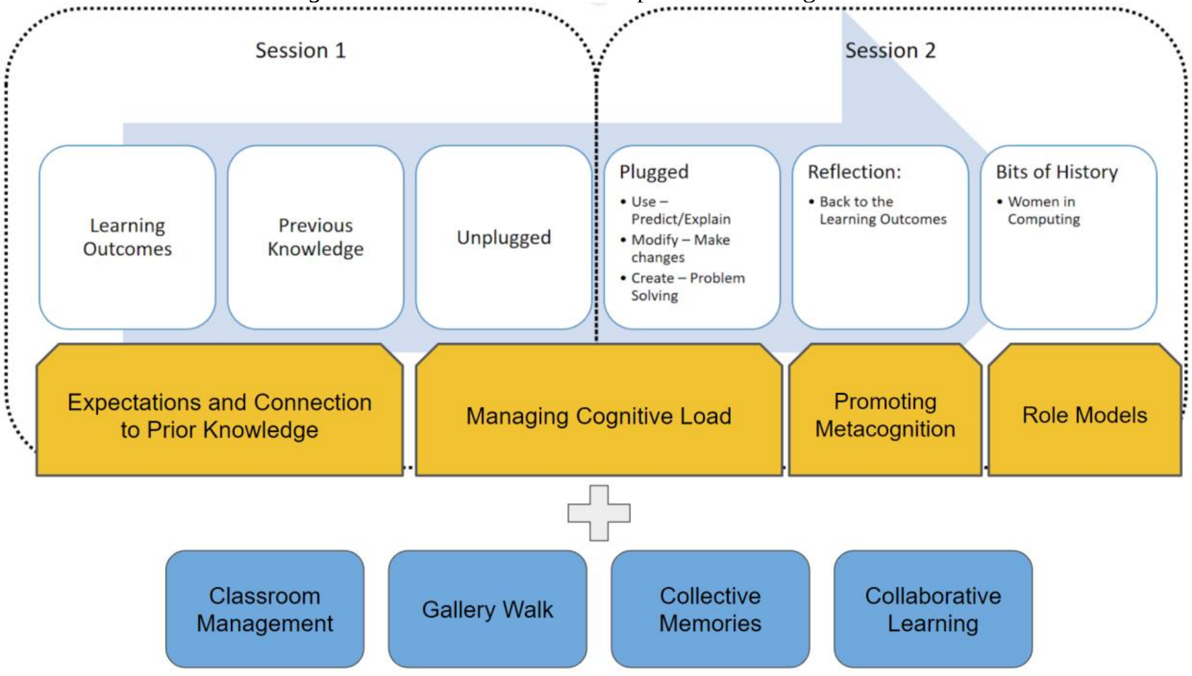
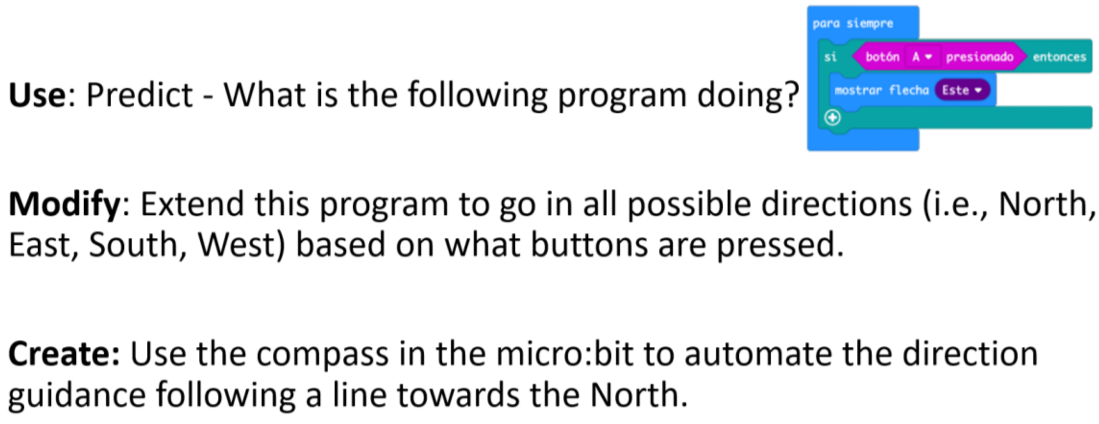


Figure 4. Sample activity of the implementation of the Use-Modify-Create progression


Use: Predict - What is the following program doing?

```
para siempre
  si botón A presionado entonces
    mostrar flecha Este
```

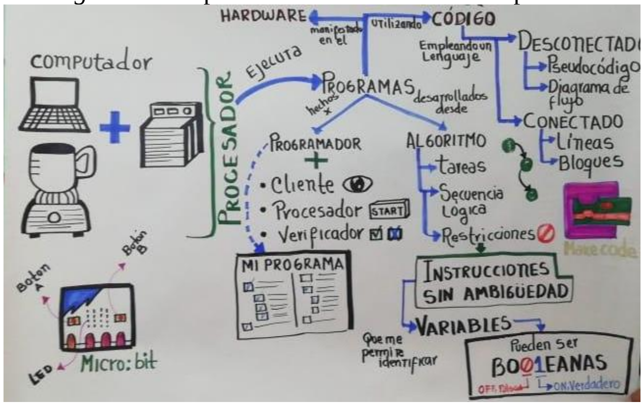
Modify: Extend this program to go in all possible directions (i.e., North, East, South, West) based on what buttons are pressed.

Create: Use the compass in the micro:bit to automate the direction guidance following a line towards the North.

The unplugged activities and the use-modify-create progression help students manage the cognitive load by developing early schemata about the problem and the algorithm. At the end of the second session, the lesson plans include two additional elements: a reflection – connecting back to the learning outcomes – and a brief biography of a woman who worked in computing (Liben & Coyle 2014). The reflection aims at promoting student metacognitive processes such as monitoring and regulating. The goal of the brief biography was to present role models for female students, aimed at encouraging more female students to pursue STEM degrees. Sequencing activities within a learning design may contribute to student self-achievement and motivation, as they can see their progress through the outputs from each activity (Katai, 2020). We hypothesize that this sequence of activities in the lesson plan will result in student development of CT skills. Appendix A summarizes the learning outcomes for each of the lesson plans. Lesson plans five and nine represented assessment instruments for the teachers to monitor student learning, so they do not have specific learning outcomes for students to develop.

Besides the lesson plans, the program also included additional strategies to support the teaching and learning process. In Colombia, preliminary studies have demonstrated that only 65% of the time in the classroom is actually dedicated to teaching and learning, while the rest of the time is devoted to things like organizing students, social or recreational activities, or repeating instructions (MEN & The World Bank, 2012). To address this issue, the mentors modeled and made visible strategies such as setting norms and rules, promoting assertive communication between teachers and students, and promoting a growth mindset. Other strategies included the gallery walk, promoting collaborative learning, and the creation of anchor charts (e.g., Figure 5).

Figure 5. Sample anchor chart from lesson plan #1



3.2. Participants

A total of *n* = 282 lower secondary (grades 6-9) and upper secondary (grades 10-11) school teachers from 30 different municipalities in Colombia participated in the program. As part of the training, participants were required to implement the lesson plans in at least one group in lower secondary school or in upper secondary school, but some of them decided to use the activities for several groups. This resulted in approximately *n* = 20,000 students being included as the ultimate beneficiaries of the program.

The mentors scheduled their last visit to apply a computational thinking test in one group at each school. If there were multiple groups at different levels, the mentors chose an intermediate level (e.g., if the teachers implemented the activities in 6th, 7th, and 8th grade, the mentors would apply it for 7th grade). In total, 4077 lower secondary and upper secondary school students comprised the treatment group that completed a performance test measuring student CT skills.

The effect of the program on students' CT skills was assessed by means of a quasi-experimental posttest only design with a comparison group. Given the variety of schools, contexts, grade levels, and local curricula, students in the control group were chosen based on the following criteria: (1) the student's grade level, (2) that their teachers had not participated in the program, (3) that the control schools were located in the same municipalities as the participating schools, and (4) that they were not using a use-modify-create progression strategy in the classroom, though they were included if they were using a different computational thinking strategy or no strategy at all. Because there are no standard computer science or computational thinking curricula for Colombian public schools, the purpose of this quasi-experimental study is to determine whether our pedagogical approach, on average, supports student learning better than existing approaches or no approach at all. The final sample for the control group included *n* = 4898 seventh and eighth-graders. The demographic characteristics of the control group are shown in Table 1.

Table 1. Characteristics of the participating students

		Rural	Urban	Total
Treatment	Female	281	1694	1975
	Male	344	1758	2102
Control	Female	1062	1329	2391
	Male	1075	1432	2507
Total		2762	6213	8975

3.3. Data collection

The students were assessed using a multiple-choice test with 12 items, measuring skills such as abstraction, algorithmic thinking, automation, debugging, and solution validation. While there is recent work designing CT assessment instruments (e.g., Román-González et al., 2017; Werner et al., 2012), many of them are specific for a programming language, for an age group, or for a specific set of concepts, do not have a validated version for Spanish-speaking students, or are not available for open use. For this project, it was important to have programming language-independent questions in Spanish, well-aligned with the program goals, and be able to validate them locally. Our recent work showed that students in this context struggled to transfer into text-based programming languages, which mostly use English keywords (Espinal et al., 2022). The questions did not include the programming language MakeCode from Micro:bit since only the students in the treatment group were exposed to such language. Most of these questions were adapted from multiple sources (Bebras Computing Challenge, 2019; Grover et al., 2015; Tan & Venables, 2010) and included tasks such as explaining the goal of a program, tracking variables, suggesting a sequence of steps, and fixing bugs from simple flow-diagrams, pseudocode, and block-based programs. Table 2 shows the format, goal, and concept that each question assessed. This table also shows the CT skills for each item according to our operational definition, and the cognitive level of the item based on the number of correct responses. Appendix B includes all the items from this instrument.

The instrument was piloted in a sample of *n* = 171 students from non-participating schools using a cognitive lab approach (Willis et al., 1991). During a cognitive lab, participants are asked to complete a task and verbalize or "think aloud" the mental processes and thoughts occurring while solving the task. Students in the pilot sample did not participate in the intervention, but shared similar individual and educational characteristics of those participating in the study, including school grade level, age, and school characteristics. The discrimination and difficulty of each item were evaluated via item analysis (Wright, 2008). Six items with a probability of less than 20% being answered correctly and a low discrimination index were revised as a result of this analysis.

Table 2. Format, goal, and concept of the questions in the multiple-choice test

Item	Format	Goal	Concept	CT Skills	Cognitive Level
1	Blocks	Debugging	Loops	Algorithm Design, Decomposition, Abstraction, and Solution Validation	Easy
2	PseudoCode	Tracing	Conditionals	Algorithm Design; Programming and Debugging; Solution Validation	Medium
3	Descriptive PseudoCode	Debugging	Loops	Computational Problem Solving; Algorithm Design; Solution Validation	Medium
4	Descriptive PseudoCode	Debugging	Sequences	Computational Problem Solving; Algorithm Design	Difficult
5	Flow Diagrams	Explaining	Conditionals	Algorithm Design	Medium
6	PseudoCode	Debugging	Conditionals	Programming and Debugging; Solution Validation; Decomposition	Easy
7	PseudoCode	Tracing	Loops and Variables	Abstraction; Algorithm Design; Programming and Debugging; Solution Validation	Difficult
8	PseudoCode	Explaining	Loops, Conditionals and Variables	Abstraction; Algorithm Design; Decomposition; Programming and Debugging; Solution Validation	Difficult
9	Flow Diagrams	Explaining	Loops and Variables	Computational Modeling; Abstraction; Algorithm Design	Medium
10	Flow Diagrams-	Debugging	Loops	Computational Problem Solving; Algorithm Design; Solution Validation	Difficult
11	PseudoCode	Explaining	Loops, Variables, and Input/Output	Abstraction; Programming and Debugging	Medium
12	Flow Diagrams	Debugging	Variables and Conditionals	Computational Problem Solving; Computational Modeling; Decomposition	Difficult

The content validity of the instrument was evaluated with the data from cognitive labs with students and with face validity with two professional experts that contributed to the design of the lesson plans. This process allowed the researchers to improve the instrument. Specifically, one of the items did not include a correct answer, and there was an indentation problem with one of the pseudocode items. The researchers fixed these two issues in the instrument, and used it to assess student CT skills in the treatment and control groups towards the end of the academic year.

3.4. Data analysis

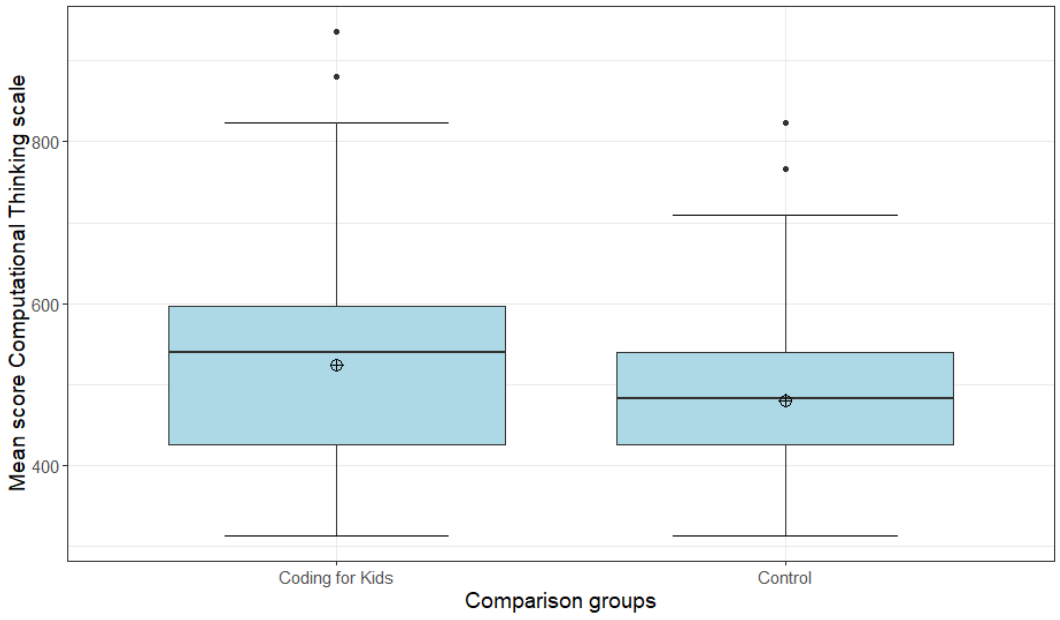
Students' responses were used to create a computational thinking scale (CTS) with a mean of 500 and a standard deviation of 100 (Streiner et al., 2015). For the construction of the scale, we first calculated the number of correct answers for each student and normalized the total of correct answers. By standardizing the scores, it is possible to compare the CT skills of the treatment group to the control group, since we can assess the position of each of the students above or below the average.

Using the CTS scores, we conducted a Welch's *t*-test to identify whether the differences in CT skills between the treatment and the control groups were statistically significant. Because the treatment and control groups have unequal sample sizes, Welch's *t*-test is preferred over the two-sample *t*-test statistic since it is more robust to violations of the assumption of variance homogeneity (Delacre et al., 2017). The Cohen's d effect size was also computed to identify how strong the effect of the program is. We used the scale suggested by Rubin (2012) to interpret Cohen's d effect size as follows: (1) Weak effect size: $|d| < 0.2$; (2) Weak to moderate: $0.2 < |d| < 0.4$; (3) Moderate: $0.40 < |d| < 0.65$; (4) Moderate to strong: $0.65 < |d| < 0.8$; (5) Strong: $0.8 < |d|$. We also used a one-way ANCOVA to see if the participants' levels of computational thinking differed significantly based on their school context (urban vs. rural) after controlling for gender. The data were examined to ensure that they met the ANCOVA assumptions. Skewness, kurtosis, and normality tests, as well as inspections of the histogram and normal Q-Q plot for computational thinking, all indicated that it was normally distributed.

4. Results

This section compared students' CT skills between the treatment and the control groups. The results show an average score for the treatment group of 525 (*SD* = 102), while the average score for the control group is 480 (*SD* = 93.9, see Figure 6). The Welch's t-test showed that the difference was statistically significant ($t(8389) = 21.4$, $p < .05$), with a moderate Cohen's d effect size of 0.45. The meaning of such effect size is that 67.4% of the participants in the treatment group performed better than the control group, and there is a 62.5% chance that a person randomly drawn from the treatment group would have a better performance than a person randomly drawn from the control group (Cumming & Calin-Jageman, 2017).

Figure 4. Coding for kids (treatment) vs. control group performance on the computational thinking scale



The one-way ANCOVA to control for gender ($F(1, 8967) = 11.418$; $p < .05$) and for school context ($F(1, 8967) = 119.189$; $p < .05$) are also statistically significant (see Figure 7 and Figure 8). Students in the treatment group showed better performance compared to students in the control group, both in urban and rural schools, and both male and female students. The mean difference between students enrolled in urban schools from the treatment and control groups is 31 points, while the difference is 58 points for rural schools.

When we analyzed the CT skills by grade level, both lower and upper secondary students in the treatment group scored considerably higher than their counterparts in the control group ($F(1, 9125) = 5.451$, $p < .05$). The mean difference between the treatment and control groups for lower secondary students is 34 points, while the difference for upper secondary students is 86 points (Figure 9).

Figure 5. Comparison of the distribution of student performance by context between coding for kids (treatment) and control groups

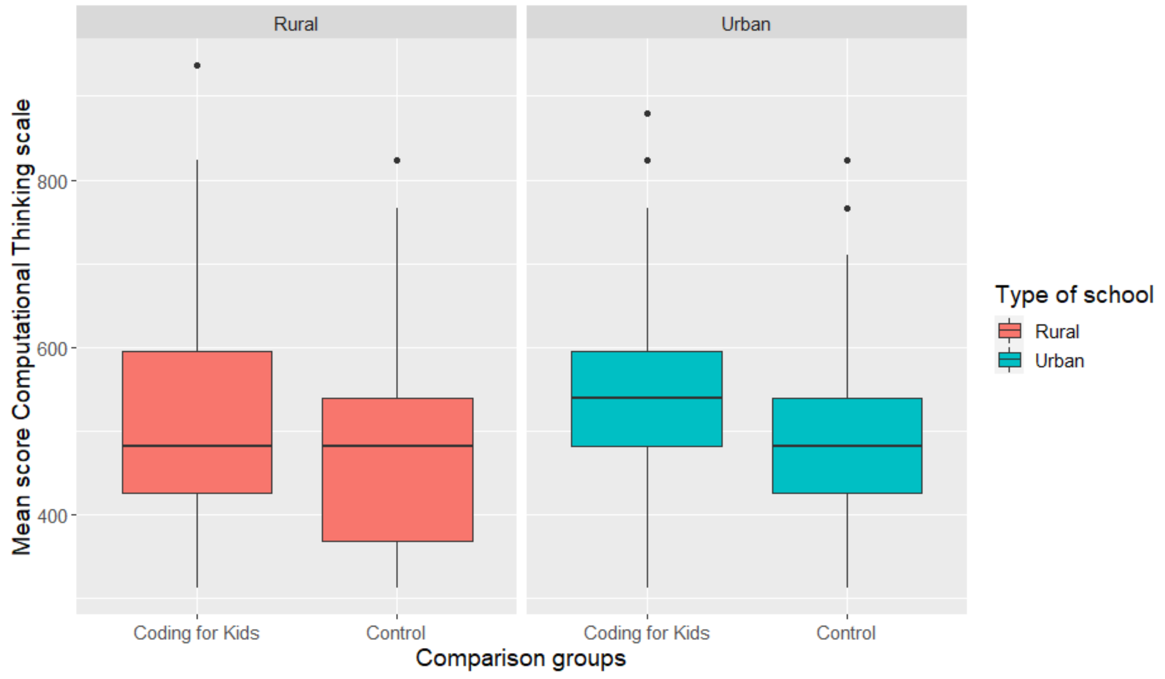


Figure 6. Comparison of the distribution of student performance by gender between Coding For Kids (Treatment) and control groups

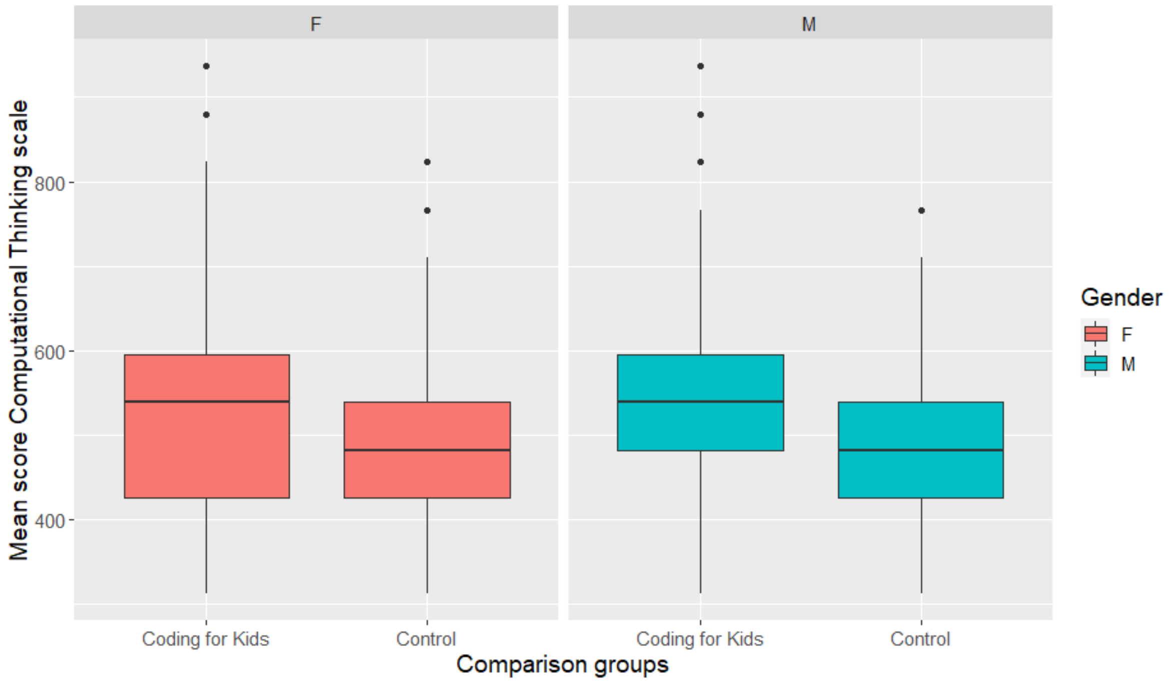
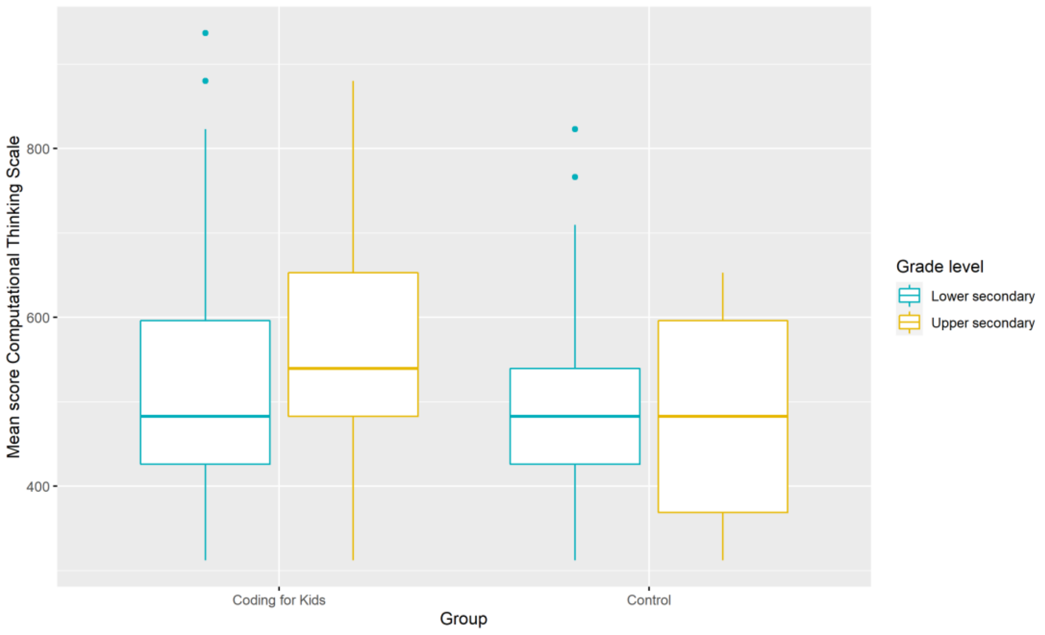


Figure 7. Comparison of student performance by grade levels between coding for kids (treatment) and control groups



5. Discussion

This study explored the *effect of the Coding for Kids project on student computational thinking knowledge and skills.* The Coding for Kids project put into action a professional development program for K-12 teachers to implement a set of lesson plans integrating computational thinking into Colombian public schools. The results from this study showed that students participating in these activities developed basic computational thinking knowledge and skills. This effect is evident when we compare participating students to a control group. Given the large scale of this study, it would be unfeasible for a single person to implement all the learning activities. While we recognize that there might be variability among teaching practices that may influence students' learning differently, our study shows that, on average, the professional development program and the learning activities have a positive effect on student learning. Although there are no pre-test data available for the experimental group, both the experimental and control groups were drawn at random from the same population. Given the large sample size, it is likely that both groups would have obtained similar results on a pre-test. On the basis of this assumption, it is reasonable to infer that the intervention significantly contributed to the higher posttest scores of the treatment group. The results were also supported by a Bayesian Mann-Whitney *U* test on independent samples (van Doorn et al., 2020), which provided strong support for the impact of the intervention. We observed a Bayes Factor (BF_{10}) of 8.906e+16, indicating a strong likelihood (8.906e+16 times in favor) of the alternative hypothesis relative to the null hypothesis. In other words, the alternative hypothesis that the students in the Coding for Kids group have higher levels of CT skills than students in the control, is strongly better at explaining the data than the null hypothesis of no difference between the two groups.

Before this project implementation, there was no specific approach to integrating CS or CT into the curriculum for Colombian public schools. Hence, the control group included any other possible approach that schools were using, including text-based programming languages, educational robotics, etc. Our results suggest that the Unplugged activities and the use-modify-create progression with a block-based programming language is, on average, more effective than other existing approaches. The effect persists when we control for context (i.e., urban and rural) and by gender. This effect is important because a large percentage of rural schools did not have access to computing devices or to an Internet connection to implement the plugged activities (i.e., approximately 70%). Thus, although the participating rural schools show greater performance variability than the urban ones, the unplugged learning activities still showed a significant effect for these schools with limited technological infrastructure. However, this effect does not mean that rural schools do not need access to the Internet or technological devices. Students from urban schools in the experimental group still showed a better performance than those from rural schools, suggesting that applying these concepts and skills using a computing device further supports student learning. Together, these results suggest that the use of unplugged learning activities and the use-modify-create progression (Lee et al., 2011; Lytle et al., 2019) may scaffold student learning of computational thinking knowledge and skills using the Micro:bit. Unplugged learning activities have become popular in recent years to support student learning without distractors and extraneous loads from the device and the programming language, promote collaborative learning, and attract more students to computer science (Bell et al., 2009, Brackmann et al., 2017). These activities help develop student understanding of basic concepts and have been suggested to be paired with plugged activities (Bell & Vahrenhold, 2018; Caeli & Yadav, 2020). We used these activities to introduce the algorithmic concepts before students engaged in programming the Micro:bit using MakeCode following the use-modify-create progression. Lee and colleagues (2011) proposed the use-modify-create progression as an approach to support student development of CT skills while maintaining the level of challenge for student development of schemata. A recent study showed promising results of the implementation of this progression based on teachers' reflections (Lytle et al., 2019) and student work (Franklin et al., 2020). In this study, we expand the evidence of the benefits of this progression by demonstrating a positive effect on student learning. While the results from the test show transfer to a certain degree (i.e., from the MakeCode language into text-based pseudocode), additional work is needed to identify whether these students are able to transfer their knowledge into text-based professional programming languages (Weintrop & Wilensky, 2017), and what challenges they face in this process. For example, a recent study showed that students are able to transfer between block-based programming languages, but they may struggle to transfer into programming languages such as Python (Espinal et al., 2022). The main difficulty seems to be that students do not understand the keywords in English when they learn using blocks in Spanish.

This study also advances our understanding of the challenges to integrating computational thinking in developing countries like Colombia. There are unique challenges in countries like Colombia for these activities, including the limited school infrastructure as well as the classroom management issues identified by previous studies (MEN & The World Bank, 2012). In a closing workshop at the end of the program, the participating mentors and teachers discussed several challenges they faced in this process. First, the limited technological infrastructure in the schools, particularly rural schools, made the program difficult to implement the plugged activities in such contexts. These teachers discussed issues such as "the lack of an internet connection in the school" and "the limited number of micro:bit devices" as challenges to the effective implementation of the lesson plans. Second, the program started in the middle of the academic year. This means that teachers already had a plan for the academic year, and they had to make adjustments, as they got involved in the professional development program. One of the participating teachers explained that the main challenge was "the time, because everything went so fast. Mostly, because we have several activities for this last part of the year." Yet another teacher highlighted that "the project is very good and it helps promote student interest, as well as developing student logical, computational, and creative thinking, but it needs to be introduced from the beginning of the academic year, so we can integrate it into the annual plan." Third, some classroom management issues persist, making classroom implementations inefficient. The participating teachers mentioned that even though they established norms, students do not always follow them: "the norms and instructions were not followed by some of the students." These challenges reflect that most of the teachers (70%) could only complete lesson plans one through five but did not have enough time to complete the rest of the lesson plans. The mentors, who conducted classroom observations and provided feedback to the participating teachers, ensured that the lesson plans were implemented following the design of the learning progression. Despite these challenges, this study showed how to operationalize unplugged activities and the use-modify-create progression into a set of lesson plans for a national professional development program, and identified a significant improvement on student CT skills when compared to a control group.

6. Conclusions, limitations, and next steps

This study explored the effect of the Coding for Kids program on developing students' CT skills. The program trained 282 k-12 teachers to integrate a set of lesson plans, aimed at developing student computational thinking in 30 different cities/areas in the country. Each lesson plan was designed for two sessions. The first session included a set of unplugged learning activities to prepare students for the second session. During the second session, students worked on the Micro:bit board following a use-modify-create progression designed to scaffold their learning process.

We created a computational thinking scale and compared participating students' performance to students in a control group (i.e., schools in similar contexts but that did not participate in the professional development program). Participating students outperformed students in a control group, both in urban and rural contexts. There are, however, some challenges involved in this process. We did not measure the effects of reducing the students' cognitive load to support student learning. Moreover, even if participating students performed better than non-participating students, the average number of correct responses in the test was low (i.e., 3.8 out of 12). The most difficult questions were those where students needed to transfer their learning about loops, a difficult concept in computer programming. There are several reasons to explain this phenomenon, including the limited infrastructure and project timing. Furthermore, the study only used a posttest to identify the possible effects of the program on student learning. Hence, any analysis of the significant differences that we found should consider these limitations. Future work will address these challenges by assessing cognitive loads and further controlling our experimental design by collecting a baseline for both experimental and control groups.

Acknowledgement

This work was funded by the Ministerio de TIC (Colombia), Computadores para Educar, and the British Council under the program Coding for Kids. The content is solely the responsibility of the authors and does not necessarily represent the official views of the supporting agencies.

References

Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., & Zagami, J. (2016). A K-6 computational thinking curriculum framework: Implications for teacher knowledge. *Educational Technology & Society*, *19*(3), 47-57.

Aranda, G., & Ferguson, J. P. (2018). Unplugged programming: The Future of teaching computational thinking? *Pedagogika*, *68*(3), 279-292. https://doi.org/10.14712/23362189.2018.859

Atkinson, R. K., Derry, S. J., Renkl, A., & Wortham, D. (2000). Learning from examples: Instructional principles from the worked examples research. *Review of educational research, 70*(2), 181-214. https://doi.org/10.3102/00346543070002181

Barr, D., Harrison, J., & Conery, L. (2011). Computational thinking: A Digital age skill for everyone. *Learning & Leading with Technology*, *38*(6), 20-23.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads*, *2*(1), 48-54. https://doi.org/10.1145/1929887.1929905

Bebras Computing Challenge. (2019). *Bebras computing challenge booklets*. http://www.bebras.uk/

Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology*, *13*(1), 20-29.

Brackmann, C. P., Román-González, M., Robles, G., Moreno-León, J., Casali, A., & Barone, D. (2017). Development of computational thinking skills through unplugged activities in primary school. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education* (pp. 65-72). ACM. https://doi.org/10.1145/3137065.3137069

Bell, T., & Vahrenhold, J. (2018). CS unplugged—How is it used, and does it work? In *Adventures between lower bounds and higher altitudes* (pp. 497-521). Springer, Cham. https://doi.org/10.1007/978-3-319-98355-4_29

Ben-Ari, M. (1998). Constructivism in computer science education. *ACM SIGCSE Bulletin*, *30*(1), 257-261. https://doi.org/10.1145/274790.274308

Caeli, E. N., & Bundsgaard, J. (2020). Computational thinking in compulsory education: A Survey study on initiatives and conceptions. *Educational Technology Research and Development*, *68*(1), 551-573. https://doi.org/10.1007/s11423-019-09694-z

Caeli, E. N., & Yadav, A. (2020). Unplugged approaches to computational thinking: A Historical perspective. *TechTrends*, *64*(1), 29-36. https://doi.org/10.1007/s11528-019-00410-5

Cumming, G., & Calin-Jageman, R. (2017). *Introduction to the new statistics: Estimation, open science, and beyond*. Routledge, Taylor & Francis Group. https://doi.org/10.4324/9781315708607

Curzon, P., Bell, T., Waite, J., & Dorling, M. (2019). Computational thinking. In *The Cambridge Handbook of Computing Education Research* (pp. 513-546). Cambridge University Press. https://doi.org/10.1017/9781108654555

Curzon, P., Dorling, M., Ng, T., Selby, C., & Woollard, J. (2014). Developing computational thinking in the classroom: A Framework. Computing at School. http://eprints.soton.ac.uk/id/eprint/369594

Delacre, M., Lakens, D., and Leys, C. (2017). Why Psychologists should by default use Welch's t-test instead of Student's t-test. *International Review of Social Psychology, 30*(1), 92–101. https://doi.org/10.5334/irsp.82

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM, 60*(6), 33–39. https://doi.org/10.1145/2998438

Espinal, A., Vieira, C., & Guerrero-Bequis, V. (2022). Student ability and difficulties with transfer from a block-based programming language into other programming languages: A Case study in Colombia. *Computer Science Education,* 1-33. https://doi.org/10.1080/08993408.2022.2079867

Faber, H. H., Wierdsma, M. D., Doornbos, R. P., van der Ven, J. S., & de Vette, K. (2017). Teaching computational thinking to primary school students via unplugged programming lessons. *Journal of the European Teacher Education Network, 12,* 13-24.

Franklin, D., Coenraad, M., Palmer, J., Eatinger, D., Zipp, A., Anaya, M., White, M., Pham, H., Gökdemir, O., & Weintrop, D. (2020). An Analysis of use-modify-create pedagogical approach's success in balancing structure and student agency. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (pp. 14-24). https://doi.org/10.1145/3372782.3406256

Grover, S. (2017). Assessing algorithmic and computational thinking in K-12: Lessons from a middle school classroom. In *Emerging research, practice, and policy on computational thinking* (pp. 269-288). Springer, Cham. https://doi.org/10.1007/978-3-319-52691-1_17

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education, 25*(2), 199-237. https://doi.org/10.1080/08993408.2015.1033142

Instituto Colombiano para el Fomento de la Educación Superior (ICFES). (2020). *Informe Nacional de Resultados para Colombia - PISA 2018* [National Results Report for Colombia - PISA 2018]. ICFES. https://www2.icfes.gov.co/documents/39286/1125661/Informe_nacional_resultados_PISA_2018.pdf/

Jonassen, D. H., Peck, K. L., & Wilson, B. G. (1999). *Learning with technology: A Constructivist perspective*. Merrill.

Jocius, R., O'Byrne, W. I., Albert, J., Joshi, D., Robinson, R., & Andrews, A. (2021). Infusing computational thinking into STEM teaching. *Educational Technology & Society, 24*(4), 166-179.

Katai, Z. (2020). Promoting computational thinking of both sciences-and humanities-oriented students: an instructional and motivational design perspective. *Educational Technology Research and Development, 68*(5), 2239-2261. https://doi.org/10.1007/s11423-020-09766-5

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., & Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads, 2*(1), 32-37. https://doi.org/10.1145/1929887.1929902

Lytle, N., Cateté, V., Boulden, D., Dong, Y., Houchins, J., Milliken, A., Isvik, A., Bounajim, D., Wiebe, E., & Barnes, T. (2019). Use, modify, create: Comparing computational thinking lesson progressions for STEM Classes. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 395-401). https://doi.org/10.1145/3304221.3319786

MEN, & The World Bank. (2012). Reporte del uso del tiempo en el aula: Evidencia para Colombia utilizando el método de observación de stallings. [Report on time use in the classroom: Evidence for Colombia using the stallings observation method].

Mohaghegh, D. M., & McCauley, M. (2016). Computational thinking: The Skill set of the 21st century. *International Journal of Computer Science and Information Technologies, 7*(3), 1524-1530.

Morrison, B. B., Margulieux, L. E., & Guzdial, M. (2015). Subgoals, context, and worked examples in learning computing problem solving. In *Proceedings of the eleventh annual international conference on international computing education research* (pp. 21-29). https://doi.org/10.1145/2787622.2787733

Mouza, C., Pan, Y. C., Yang, H., & Pollock, L. (2020). A Multiyear investigation of student computational thinking concepts, practices, and perspectives in an after-school computing program. *Journal of Educational Computing Research, 58*(5), 1029-1056. https://doi.org/10.1177/0735633120905605

Mselle, L. J., & Twaakyondo, H. (2012). The Impact of Memory Transfer Language (MTL) on reducing misconceptions in teaching programming to novices. *International Journal of Machine Learning and Applications, 1*(1), 6. https://doi.org/10.4102/ijmla.v1i1.3

Mueller, J., Beckett, D., Hennessey, E., & Shodiev, H. (2017). Assessing computational thinking across the curriculum. In *Emerging research, practice, and policy on computational thinking* (pp. 251-267). Springer, Cham. https://doi.org/10.1007/978-3-319-52691-1_16

Noh, J., & Lee, J. (2020). Effects of robotics programming on the computational thinking and creativity of elementary school students. *Educational Technology Research and Development, 68*(1), 463-484. https://doi.org/10.1007/s11423-019-09708-w

Organisation for Economic Co-operation and Development (OECD). (2020). *Country note: Colombia*. https://www.oecd.org/pisa/publications/PISA2018_CN_COL_ESP.pdf

Robins, A. V. (2019). Novice programmers and introductory programming. In *The Cambridge Handbook of Computing Education Research* (pp. 327-376). Cambridge University Press. https://doi.org/10.1017/9781108654555

Royal Society. (2012). *Shut down or restart?: The Way forward for computing in UK schools*. Royal Society. https://royalsociety.org/~/media/education/computing-in-schools/2012-01-12-computing-in-schools.pdf

Román-González, M., Pérez-González, J. C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in human behavior*, *72*, 678-691. https://doi.org/10.1016/j.chb.2016.08.047

Rubin, A. (2012). *Statistics for evidence-based practice and evaluation*. Cengage Learning.

Sanford, J. F., & Naidu, J. T. (2016). Computational thinking concepts for grade school. *Contemporary Issues in Education Research*, *9*(1), 23-32. https://doi.org/10.19030/cier.v9i1.9547

Sentance, S., Waite, J., Yeomans, L., & MacLeod, E. (2017). Teaching with physical computing devices: The BBC micro: bit initiative. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education* (pp. 87-96). https://doi.org/10.1145/3137065.3137083

Streiner, D. L., Norman, G. R., y Cairney, J. (2015). *Health measurement scales: A Practical guide to their development and use* (Fifth edition). Oxford University Press. https://doi.org/10.1093/med/9780199685219.001.0001

Sweller, J., van Merriënboer, J. J., & Paas, F. (2019). Cognitive architecture and instructional design: 20 years later. *Educational Psychology Review*, 1-32. https://doi.org/10.1007/s10648-019-09465-5

Tan, G., & Venables, A. (2010). Wearing the assessment 'BRACElet'. *Journal of Information Technology Education: Innovations in Practice*, *9*(1), 25-34. https://www.learntechlib.org/p/111692/

Tricot, A., (2017). Les contraintes spécifiques des apprentissages scolaires [The Specific constraints of school learning]. *Psychologie & Education*, *2017-1*, 68-82. https://hal.science/hal-01628833

van Doorn, J., Ly, A., Marsman, M., and Wagenmakers, E.-J. (2020). Bayesian rank-based hypothesis testing for the rank sum test, the signed rank test, and Spearman's ρ. *Journal of Applied Statistics*, *47*(16), 2984–3006. https://doi.org/10.1080/02664763.2019.1709053

Vieira, C., Magana, A. J., Falk, M. L., & Garcia, R. E. (2017). Writing in-code comments to self-explain in computational science and engineering education. *ACM Transactions on Computing Education (TOCE)*, *17*(4), 1-21. https://doi.org/10.1145/3058751

Vieira, C., Magana, A. J., Roy, A., & Falk, M. L. (2019). Student explanations in the context of computational science and engineering education. *Cognition and Instruction*, *37*(2), 201-231. https://doi.org/10.1080/07370008.2018.1539738

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. https://doi.org/10.1007/s10956-015-9581-5

Weintrop, D., & Wilensky, U. (2017). Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE)*, *18*(1), 1-25. https://doi.org/10.1145/3089799

Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012). The Fairy performance assessment: Measuring computational thinking in middle school. In *Proceedings of the 43rd ACM technical symposium on computer science education* (pp. 215e220). http://dx.doi.org/10.1145/2157136.2157200

Willis, G. B., Royston, P., & Bercini, D. (1991). The Use of verbal report methods in the development and testing of survey questionnaires. *Applied Cognitive Psychology*, *5*(3), 251–267. https://doi.org/10.1002/acp.2350050307

Wing, J. (2011) Research notebook: Computational thinking - What and why? In *The Link Magazine* (pp. 20-23). Carneige Mellon

Wing, J. M., & Stanzione, D. (2016). Progress in computational thinking, and expanding the HPC community. *Communications of the ACM*. https://doi.org/10.1145/2933410

Wright, R. (2008). *Educational assessment: Tests and measurements in the age of accountability*. SAGE. https://dx.doi.org/10.4135/9781483329673

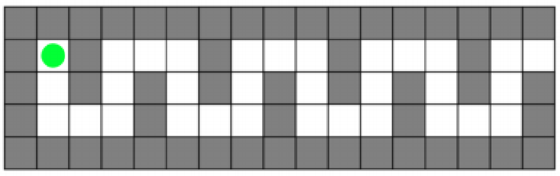
Yadav, A., Stephenson, C., & Hong, H. (2017). Computational thinking for teacher education. *Communications of the ACM*, *60*(4), 55-62. https://doi.org/10.1145/2994591

Appendix A – Learning outcomes of the lesson plans

Lesson plan	Learning goals
1	• Identify and write a set of steps and instructions to carry out a task. [Algorithm Design] • Simulate the execution of this set of instructions and steps. [Decomposition; Solution Validation] • Use the MakeCode editor of the Micro:bit to write a program and simulate. [Programming and Debugging] • Use inputs and outputs of the Micro:bit [Programming and Debugging] • Use Boolean variables. [Algorithm Design] • Describe what is a program, a programmer and a processor, an input and an output. [Programming and Debugging]
2	• Use loops to repeat a set of actions [Abstraction] • Recognize that a loop can be repeated indefinitely, a number of times or as long as a condition is met or not. [Abstraction; Algorithm Design] • Interpret and create flow diagrams. [Algorithm Design] • Load the code into the Micro: bit and verify the operation of the program [Programming and Debugging] • Use continuous input variables. [Programming and Debugging] • Show a variable like the temperature in the array of LEDs. [Programming and Debugging; Solution Validation]
3	• Use boolean input variables [Programming and Debugging] • Communicate instructions using the LED screen [Programming and Debugging] • Interpret a sequence of instructions and a flow diagram to solve a problem such as a maze. [Algorithm Design] • Use logical operations [Algorithm Design; Decompositon] • Use loops that are repeated until the task is finished. [Abstraction; Decomposition; Programming and Debugging]
4	• Define an internal variable that stores a numeric value [Programming and Debugging] • Perform operations with the values in these variables [Programming and Debugging]
5	• Test N/A
6	• Structure a problem situation. [Computational Problem Solving] • Identify specifications. [Computational Problem Solving] • Identify restrictions. [Computational Problem Solving] • - Design, build and test a prototype to evaluate some principles of the solution. [Programming and Debugging; Computational Problem Solving]
7	• Collect data from the environment (e.g., temperature) using the Micro:bit [Programming and Debugging] • Use conditionals to make decisions using the data collected from the environment [Algorithm Design; Decomposition] • Compute basic statistical measures such as the mean, and the min and max values [Computational Problem Solving] • Create a program that controls the functioning of the Micro:bit using the buttons [Programming and Debugging]
8	• Simulate natural events to predict possible outcomes [Computational Modeling] • Send and receive information between Micro:bit devices. [Programming and Debugging]
9	• Test N/A

Appendix B - Computational Thinking Skills Assessment Test

1. Help the green robot to exit the maze using one of the set of instructins below. Note: The number of times that the sequence repeats itself (3 or 4 times) will start counting after it executes the for the first time. For instance, if it says “3 times”, it will be executed 4 times in total.



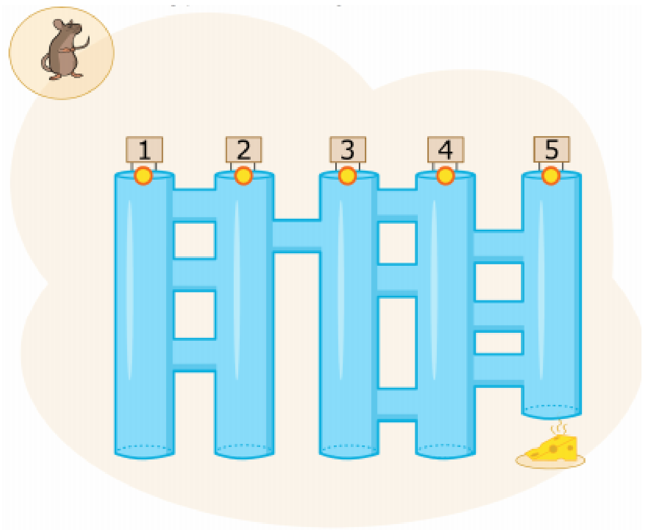
2. Consider the following program

```
if (a>b) then
    if(b<c) then
        print c
    else
        print b
else if (a<c) then
    print c
else
    print a
```

If a= 3, b= 8, and c=10, ¿what will this program print?

a. 3
b. 8
c. 10
d. 10 and 3

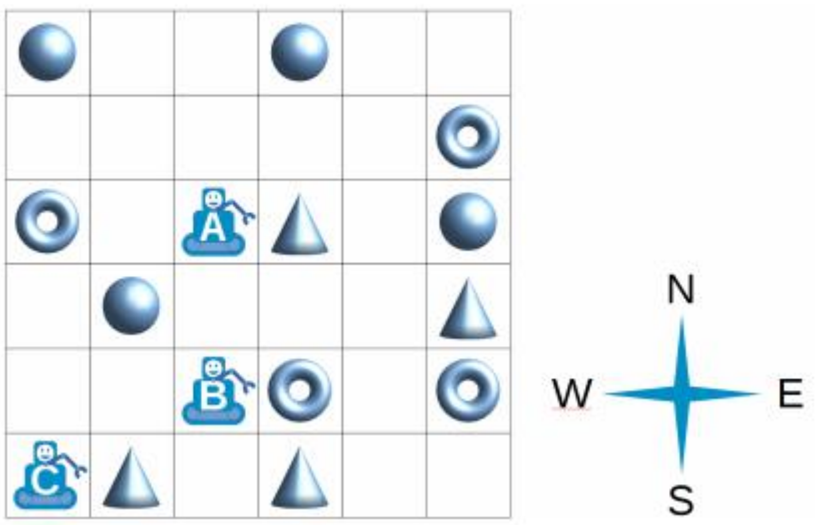
3. A mouse is at the entrance of a tube system. It wants to reach the cheese at the end of tube 5.The mouse always follows these command
(1) Go downwards until a crossing
(2) At the crossing, move through to the next vertical tube
(3) Go to command (1).
In which tube should the mouse start so that it reaches the cheese?



a. 1
b. 2
c. 3
d. 4
e. 5

4. In a warehouse, three robots always work as a team. When the team gets a direction instruction (N, S, E, W), all robots in the grid will move one square in that direction at the same time. After following a list of instructions, the robots all pick up the object found in their final square.

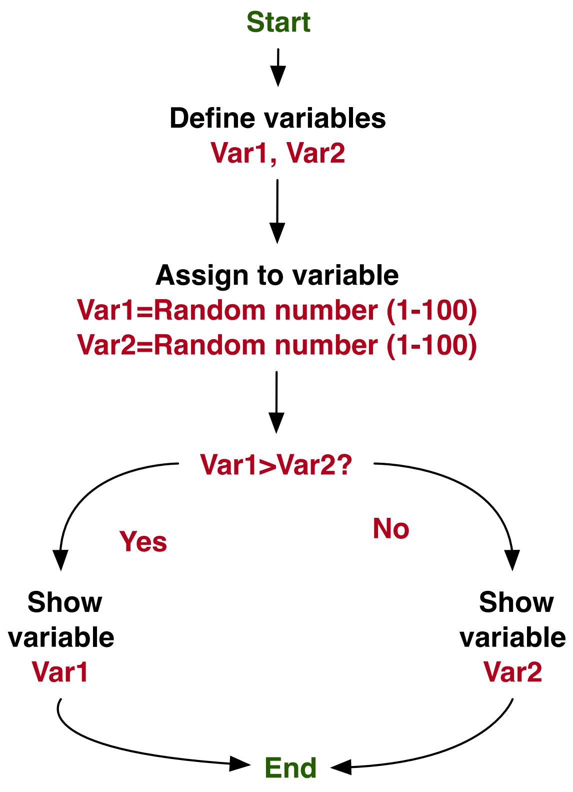
For example, if we give the list N, N, S, S, E to the team, then robot A will pick up a cone, robot B will pick up a ring, and robot C will pick up a cone



Which list of instructions can be sent to the robots so that the team picks up exactly a sphere, a cone, and a ring?

a. N, E, E, E
b. N, E, E, S, E
c. N, N, S, E, N
d. N, E, E, S, W

5. Consider the following flow diagram:



What is the goal of this algorithm?

a. Show two random numbers
b. Show the largest between two random numbers
c. Show the smallest between two random numbers
d. None of the above

6. Consider the following pseudocode, where a and b are variables:

```
a=3
b=5
if(a>2 and b<4) then
    print a
end
```

¿Will the *print a* instruction be executed?

7. Consider the following pseudocode. What will the program print at the end?

```
num=0
cont=1

while (cont<5) do
    num = num + cont
    cont = cont + 1

print num
```

a. 15
b. 0
c. 1
d. 10

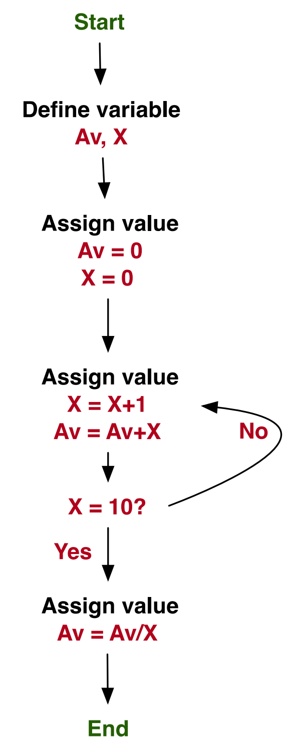
8. What is the purpose of the following program?

```
while (cont<10) do
    if cont even? then
        num = num + cont
    cont = cont + 1

print num
```

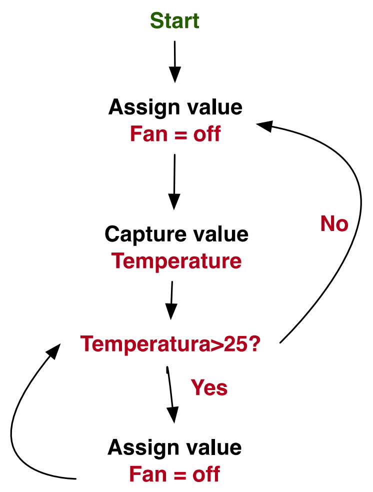
a. Creates a list including the even numbers between 1 and 10
b. Sums up all numbers from 1 to 10
c. Counts up to 10 if the numbers are even
d. Sums up all the even numbers between 1 and 9

9. What is the purpose of the following algorithm represented in a flow diagram?



a. Finds the number 10 to store it into the X variable.
b. Sums up all numbers between 1 and 10
c. Computes the average for the numbers from 1 to 10
d. Divides the Av value by all numbers from 1 to 10

10. Andrea created a flow diagram to design an algorithm that will allow her to automatically turn on the fan when her room is too hot. However, she is not sure this will work. What would you recommend?



a. The program does not work; the End (Fin) instruction is missing
b. The program does not work; she should not be turning the fan off
c. The program works correctly and she may implement it now to run in a processor
d. The program does not work; it should collect the temperature once more after turning the fan on

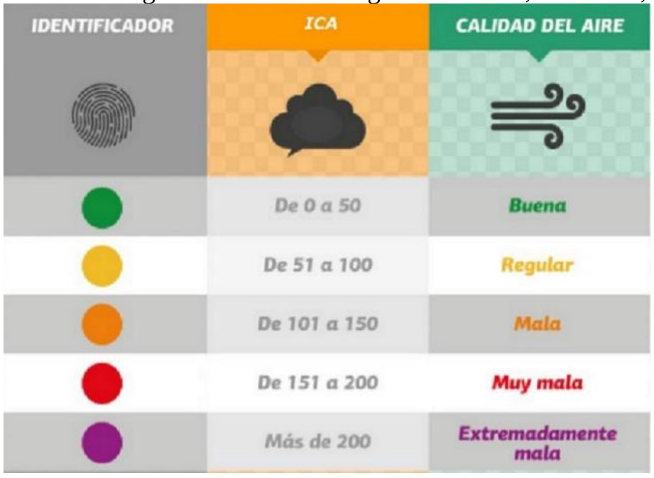
11. What is the purpose of the following program?

```
name = input('what is your name?')
cont=0

while cont<3:
    print('Hi, ')
    print(name)
    cont=cont+1
```

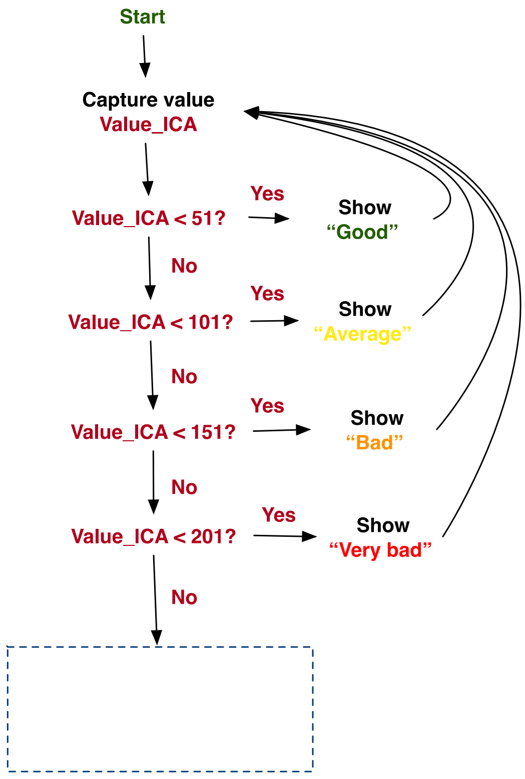
a. Asks for a name three times
b. Says hi, and print the name three times
c. Asks for a name, and says hi and prints the name twice
d. Asks for a name, and says hi and prints the name three times

12. Air pollution in the main cities is an important issue for the governors. Air pollution can be measured using the Air Quality Index (ICA, in Spanish).
The following table shows the ranges for a Good, Moderate, Bad, and Very Bad ICA



IDENTIFICADOR	ICA	CALIDAD DEL AIRE
🟢	De 0 a 50	Buena
🟡	De 51 a 100	Regular
🟠	De 101 a 150	Mala
🔴	De 151 a 200	Muy mala
🟣	Más de 200	Extremadamente mala

Nick is in charge of creating a monitoring system to represent these values, and created the following flow diagram. However, he does not know how to finish it yet. What should he include in the dotted box?



a. Ask whether <u>*valorICA*</u> is smaller than 251, show "Extremadamente mala" and go back to *Capturar*
b. Show "Extremadamente mala" and go back to *Capturar*
c. Go back to *Capturar*
d. End