Sánchez, J. (2024) Editor. Nuevas Ideas en Informática Educativa, Volumen 18, p. 134 - 141. Santiago de Chile.

First Steps in Programming with Scratch: The Impact of Block Programming on Learning Computational Logic

Maria G. A. Araújo
IFPE
Belo Jardim, Brazil
mgaa@discente.ifpe.edu.br

Ana V. R. Santos
IFPE
Belo Jardim, Brazil
avrs@discente.ifpe.edu.br

Andressa G. F. Lima
IFPE
Belo Jardim, Brazil
agfl@discente.ifpe.edu.br

Deise C. Silva
IFPE
Belo Jardim, Brazil
dcs12@discente.ifpe.edu.br

Elayne F. M. Oliveira
IFPE
Belo Jardim, Brazil
efmo@discente.ifpe.edu.br

Fábio B. Ferreira
IFPE
Belo Jardim, Brazil
fbf@discente.ifpe.edu.br

Maria R. C. Santos
IFPE
Belo Jardim, Brazil
mrcs2@discente.ifpe.edu.br

João A. Silva
IFPE
Belo Jardim, Brazil
joao.almeida@belojardim.ifpe.edu.br

ABSTRACT
In the Fourth Industrial Revolution and the rapid digital transformation, mastering technology and logical reasoning is essential for success in the job market. This study evaluates the influence of Scratch as a preparatory tool for the development of programming logic and computational thinking before learning traditional languages. Conducted at the Instituto Federal de Pernambuco - IFPE, Belo Jardim Campus, the study involved students from the extension project "Iniciação à Programação ", divided into two groups: one group completed Module I with Scratch before Module II with Python, and the other only Module II. Through pre- and post-course forms, performance and understanding of programming concepts were measured. The results indicate that students who used Scratch demonstrated a more robust understanding of programming logic. The study concludes that integrating Scratch into curricula can democratize access to programming knowledge, promoting digital inclusion and equipping students with essential skills for the digital job market.

Keywords
Scratch; Programming Logic; Computational Thinking.

ACM Classification Keywords
K.3.1 [Computers and Education]: Computer Uses in Education; K.3.2 [Computers and Education]: Computer and Information Science Education; D.1.7 [Programming Techniques]: Visual Programming.

INTRODUCTION

In the current scenario, characterized by the Fourth Industrial Revolution and accelerated digital transformation, technology is rapidly changing and the demand for skilled professionals is constantly growing [5]. This dynamic environment highlights the importance of mastering technological skills and developing logical reasoning as fundamental elements for success in the job market [5, 15]. With technology increasingly integrated into educational and professional spheres, the ability to understand and apply programming concepts has become an indispensable skill [15].

The acquisition of the ability to solve problems through computational approaches and the skill to critically evaluate solutions are crucial components of literacy in the knowledge society era [15, 3]. By learning to develop computer programs, individuals not only gain technical skills but also valuable strategies for problem-solving, designing innovative projects, and effectively communicating ideas in a structured and logical manner [15]. These competencies are fundamental for success in an increasingly digital and interconnected work environment [3].

It is believed that using the Scratch tool as a support in teaching and learning processes enables the creation of more interactive, dynamic, and intuitive lessons, promoting the development of students' logical reasoning and mastery of programming languages [3]. This approach is supported by the idea that block-based programming languages offer an intuitive and accessible approach to introduce programming concepts, preparing students for a smoother transition to textual languages [1].

Through this research, we aim to investigate whether using Scratch as a teaching tool can facilitate the development of logic and computational thinking in students with no prior programming experience. Visual programming languages, like Scratch, use code blocks that can be dragged and dropped to create programs, making it easier to understand

the underlying logic of programming without the need to memorize complex syntax.

The objective is to analyze whether this intuitive approach significantly contributes to the development of fundamental skills in programming logic and computational thinking before learning a traditional programming language. By doing so, we aim to contribute to the democratization of access to programming knowledge and digital inclusion, strengthening the essential skills needed for the job market in the digital age.

THEORETICAL FOUNDATION

In this section, an exploration of the interconnection between computational thinking and programming logic will be presented, highlighting their relevance in advancing technology and problem-solving. Following this, the distinction between traditional programming and block programming will be discussed, providing insights into how these paradigms influence the approach to teaching and learning programming. Additionally, the extension project "Iniciação à Programação" will be addressed, which aims to promote the learning of programming skills such as computational thinking and logical reasoning through the use of Scratch and Python.

Computational Thinking and Programming Logic

Computational thinking and programming logic are closely related and play a fundamental role in technological development and problem-solving. Computational thinking is a skill that involves applying computational principles to collaboratively and effectively solve problems. According to Brackmann [1], "it is the ability to solve problems clearly and logically, which can be performed by both humans and machines." This approach is interdisciplinary and can be applied in different fields of knowledge, promoting systematic problem-solving through abstraction, pattern recognition, decomposition, and algorithmic design.

The introduction of computational thinking in the early stages of education is seen as a bridge that connects different fields of knowledge, helping to demystify the inherent complexity of information technology industries [10]. Programming logic, on the other hand, is a more specific component of computational thinking. It deals with the rules and structures for creating algorithms that instruct computers to perform specific tasks, using sequences, loops, conditions, and variables [17]. Programming logic is essential for the practical application of computational thinking, transforming abstract ideas into concrete instructions that can be coded and executed by a computer [8].

The main difference between these concepts lies in their scope of application. While computational thinking is a general and interdisciplinary skill that can be applied to solve high-level problems, programming logic is the practical application of these skills in the specific field of computer programming. Computational thinking focuses on problem formulation and determining the best strategies to solve them. However, programming logic involves implementing these strategies so that computers can understand and execute them [11].

Given the importance of this knowledge, namely the distinction between computational thinking and programming logic as discussed earlier, there is a need to create technologies that enable children, teenagers, and adults to learn computational thinking and programming logic. Incorporating these skills into the curriculum from the beginning, through tools such as Scratch, Portugol, and Visualg, is essential to prepare future generations to solve challenges in a productive and creative manner. For instance, Scratch provides a simple and intuitive interface that facilitates learning computational thinking and programming logic concepts for children and teenagers [16].

In short, computational thinking and programming logic are fundamental skills for developing individuals capable of tackling the challenges of the digital age. Acquiring these skills from an early age can significantly contribute to shaping a society that is more prepared and innovative, capable of effectively using technology to solve problems across various fields of knowledge [18].

Traditional Programming and Block Programming

Traditional programming, also known as procedural or imperative programming, is one of the most well-known programming paradigms today and was one of the earliest paradigms adopted. In Imperative Programming, the focus is on how the program should operate, similar to the internal functioning of a computer that alters the state of memory while executing instructions [13]. All the necessary rules to achieve a specific outcome are explicitly written [9]. In this way, a sequence of instructions is executed line by line to achieve the desired objective.

For the operation of this paradigm, we need to use a programming language such as Java, JavaScript, or Python [13]. These languages direct the entire flow of the program through structures such as conditionals, loops, and functions. Data is manipulated through assignment statements and logical operators, ensuring controlled and precise execution [13].

In turn, the paradigm of block programming emerged to aid in the transition from purely abstract knowledge to something more 'concrete' [7]. Inspired by LEGO blocks, this paradigm allows the construction of programs using blocks of commands that connect to each other through a graphical interface [7]. These blocks represent a series of commands that will compose the program's execution flow according to the desired outcome. Due to its attractive and accessible interface, block programming proves to be an excellent option for initiating the study of programming. The color and shape of the commands, the organization of the blocks, and the program construction mechanism are easily navigable and displayed to support novice students in writing programs [2].

Scratch is a popular tool that uses the block programming paradigm. Developed by the MIT Media Lab, Scratch simplifies programming for beginners by allowing them to create programs by dragging and connecting code blocks through a graphical interface. This visual approach helps to transform abstract concepts into something more tangible and understandable. Scratch teaches programming logic and computational thinking in a fun and accessible way, making it a valuable tool for educators and students [8].

Project Iniciação à Programação

The extension project titled "Iniciação à Programação" aims to promote the participation of individuals interested in acquiring programming skills such as computational thinking and logical reasoning. It was noted that the majority of students were not familiar with the fundamental concepts of programming and had not been exposed to this knowledge during their school education.

In order to facilitate the understanding of the topics covered by students, the course is divided into two modules. In the first stage, Scratch is used as a tool to teach the essential fundamentals of computational logical thinking and its application in practical situations. In the second part, the same concepts introduced in the first stage, with a higher level of difficulty, are conveyed through the Python textual programming language. With this approach, the primary aim was to demystify the idea that developing computational thinking was something complex and inaccessible to all interested parties. Thus, the goal was to spark interest and enhance the existing abilities of the students.

RELATED WORKS

Several studies have investigated the use of Scratch as a tool for developing computational thinking and programming logic, addressing its application in various educational contexts, such as elementary and secondary schools, and in teaching areas like mathematics and physics. Souza et al. [12] report how Scratch has been used in these contexts to introduce fundamental programming concepts in a visual and interactive manner, facilitating the development of computational thinking.

Monteiro and Holanda [6] investigated the use of Scratch in a public school to develop computational thinking. By employing active learning methodologies, the study demonstrated that Scratch facilitated the understanding of fundamental concepts and increased student engagement. The authors highlighted the importance of a continuous project to maintain student interest and noted challenges related to laboratory infrastructure and remote teaching. These conclusions reinforce that Scratch is an accessible and interactive tool for secondary education.

Stewart and Baek [14] reviewed the literature on the use of Scratch in elementary education and identified five main themes: tangible blocks, integration across various subjects, games for developing computational thinking, assessment through projects, and methods that influence the teaching of computational thinking. The review highlights how Scratch facilitates the understanding of complex concepts and adapts to different educational contexts, promoting the development of computational thinking from an early age.

Additionally, Herawati et al. [4] analyzed the use of Scratch in teaching physics and found that it enhances computational thinking by making abstract concepts more accessible and engaging. The study showed that Scratch facilitates a concrete understanding of physics and develops analytical skills such as decomposition and pattern recognition. These results highlight Scratch's potential as a valuable resource for more integrated learning of complex concepts.

While these studies highlight the benefits of Scratch, there is a gap in the literature regarding a direct comparison between the effectiveness of Scratch and the learning of complex textual languages in developing programming logic and computational thinking. This study aims to fill this gap by comparing students who learned Scratch before a textual language with those who learned a complex language directly, providing insights into building a solid foundation in programming and computational thinking.

METHODOLOGICAL PROCEDURES

In this section, all methodological procedures will be presented, including the research plan, information about the participants, data collection, and analysis. This research had an applied nature with an empirical approach and quantitative methodology, utilizing a case study strategy and a deductive scientific method. The technical procedures involved analyzing student performance through forms and statistical analysis. Data collection was conducted through pre and post-course forms, recording students' responses for subsequent statistical analysis.

Research Plan

The case study discussed in this work involved two experimental groups of students. The students participated in Module II of the "Iniciação à Programação" Project, held at the Instituto Federal de Ciência e Tecnologia, located on the Belo Jardim campus, a city in the interior of Pernambuco, Brazil.

The research was conducted to evaluate how prior knowledge of programming with a block-based language influences participants' foundational programming logic and computational thinking before transitioning to a conventional textual language. The data analysis compared the performance of groups on fundamental programming concepts to determine whether experience with the block-based language resulted in a stronger foundation in logic and computational thinking.

Participants

This study, designed as a case study, involved the participation of 15 students, aged between 16 and 46 years old. As a requirement, all participants were required to have completed elementary school. The students were divided into two groups.

- Group A: Seven (7) participants in Module II who previously completed Module I previously, meaning they had prior knowledge gained from studying basic concepts with block programming.
- Group B: Eight (8) participants in Module II did not take part in Module I, meaning they did not have prior knowledge.

Procedures

The methodology used during the classes was designed to balance theory and practice, aiming to facilitate learning. At the end of each module, students had the opportunity to work on individual projects and then on group projects. They could choose a problem of interest to them and solve it through programming, applying practically what they learned during the classes. Module I lasted for 9 weeks, while Module II extended over 13 weeks.

It is important to highlight that the scheduled content for both modules was similar, with Module II incorporating more advanced topics. This addition aimed to deepen students' knowledge, building on the foundation acquired in Module I.

Module I of the Extension Course

Module I of the project lasted for three months, with evening classes held once a week, each lasting 3 hours. The topics covered during the classes were detailed in Table 1.

Lesson	Content
1	Introductory class
2	Computational thinking
3	Variables, constants, data types, and logical and mathematical operators
4	Conditional structures
5	Loop structures
6	Conditional structures + Loop structures
7	Individual practical project
8	Group practical project
9	Presentation of group practical projects

Table 1. Contents of Module I.

Module II of the Extension Course

The students in this module actively participated in 13 in-person classes, held twice a week in the evenings, from September 13 to October 30, 2023. The contents planned by the instructors to be covered during the classes are detailed in Table 2.

Lesson	Content
1	Course presentation and algorithm
2	Computational thinking and flowchart
3	Constants, variables, and data types in Python
4	Arithmetic, comparison and logical operators
5	Conditional structures
6	Loop structures
7	Conditional structures + Loop structures
8	Arrays
9	Functions
10	String manipulation
11	Individual practical project
12	Group practical project
13	Presentation of group practical projects

Table 2. Contents of Module II.

Data Collection

For data collection, a set of questionnaires was administered at two distinct times: at the beginning (pre-course) and at the end (post-course) of Module II. These questionnaires were essential to measure the participants' progress and understanding throughout the course. The Google Forms platform was used for administering the forms. The administration of the forms occurred as follows: all participants received a pre-course questionnaire before starting Module II, and a post-course questionnaire immediately after completing the module. The timing of administration varied according to the module's duration, with the pre-course questionnaire administered on the first day of classes and the post-course questionnaire on the last day.

We ensured strict anonymity and confidentiality of the collected data. The data were securely stored, accessible only to the researchers involved in the study. The forms were structured with multiple-choice questions, focusing on various aspects of participants' learning and perception of programming. In Table 3, we present the question prompts without including the answer choices.

ID	Question
Q01	What do you believe programming logic to be?
Q02	What do you believe an algorithm to be?
Q03	What do you believe a programming language to be?
Q04	What do you believe a variable to be?
Q05	What do you believe a constant to be?
Q06	Which value below is not a predefined constant?
Q07	What do you believe computational thinking to be?

ID	Question
Q08	What do you believe a conditional structure to be?
Q09	What are the types of conditional structures?
Q10	What are loops used for in programming?
Q11	Which of the following examples combines the conditional structure and the loop structure?
Q12	Identify the option that presents a typical example of Boolean data.
Q13	Regarding loops, select the option that presents the structure which is a definite loop, meaning it is used when the number of times the commands should be executed is known in advance — before the loop begins.

Table 3. Questions used for knowledge assessment.

These questions aimed to capture the participants' perceptions of learning programming, their difficulties and progress, and the effectiveness of the methodology employed. The analysis of the responses provided a comprehensive understanding of the impact of block-based programming on the process of developing logic and computational thinking before learning more complex programming languages.

Data Analysis

This research adopted a quantitative approach, as it required a comprehensive study of the object of investigation, taking into account the school context in which it is embedded and the characteristics of the society to which it belongs. Conversely, the subjective nature of qualitative research makes fieldwork indispensable.

Additionally, a comparative analysis was conducted between the pre-course and post-course questionnaire results. This comparison was crucial to assess the participants' progress and the effectiveness of the applied methodology. The difference in responses between the two questionnaire administrations provided insights into the course's impact on participants' understanding and confidence in programming.

To ensure the robustness of the analysis, all data were reviewed and verified to eliminate any inconsistencies or invalid responses. This way, we ensured that the results presented accurately reflected the participants' experience and learning throughout the course.

The results of the quantitative analysis were then interpreted in light of the study's objectives, allowing for well-founded conclusions about the effectiveness of block-based programming as an educational tool to introduce children, adolescents, and adults to the world of programming.

ANALYSIS OF RESULTS

Upon initial evaluation of the data obtained, we observed in Table 4 a significant change in the percentage of correct answers by students (Group A and Group B) after completing Module II of the project compared to before classes began, indicating the extent of progress achieved with the taught lesson.

Question	Correct answers		Errors	
	Pre-course	Post-course	Pre-course	Post-course
Q01	100%	100%	0%	0%
Q02	80%	86,70%	20%	13,30%
Q03	100%	100%	0%	0%
Q04	93,30%	100%	6,70%	0%
Q05	80%	93,30%	20%	6,70%
Q06	60%	73,30%	40%	26,70%
Q07	86,70%	93,30%	13,30%	6,70%
Q08	80%	86,70%	20%	13,30%
Q09	46,70%	80%	53,30%	20%
Q10	86,70%	93,30%	13,30%	6,70%
Q11	40%	80%	60%	20%
Q12	66,70%	86,70%	33,30%	13,30%
Q13	60%	66,70%	40%	33,30%
Average	75%	88%	25%	12%

Table 4. Performance of pre and post Module II questions of the project.

Upon analyzing the table above, it can be observed that the average percentage of correct answers after the course is 13% higher than the average of correct answers before the course, indicating a significant increase in the students' average performance. Table 4 also shows the performance on specific questions and the limitations of students in certain subjects.

When comparing the pre-course and post-course assessments, we noticed an increase in the number of questions with 100% correct answers in the post-course assessment, totaling three questions, one more than in the pre-course assessment. However, in the pre-course assessment, four questions had between 40% and 60% correct answers, while in the post-course assessment, there were no questions in this range. Instead, the lowest percentage of correct answers in the post-course assessment was 66.7%.

In the pre-course form, questions Q09 and Q11 stand out, which are related to conditional and loop structures. These questions show a low performance in correct answers, with less than 60% of students answering correctly. However, in the post-course assessment, these numbers increased significantly, with 80% of students answering these questions correctly.

Results from Group A

Analyzing Group A separately, it is observed that their data are predominantly positive. The evaluated students in the group demonstrate a good command of the theoretical aspects of what programming is and its development methods. This is evidenced by Figure 1, which shows excellent performance in the questions indicating a higher number of correct answers.

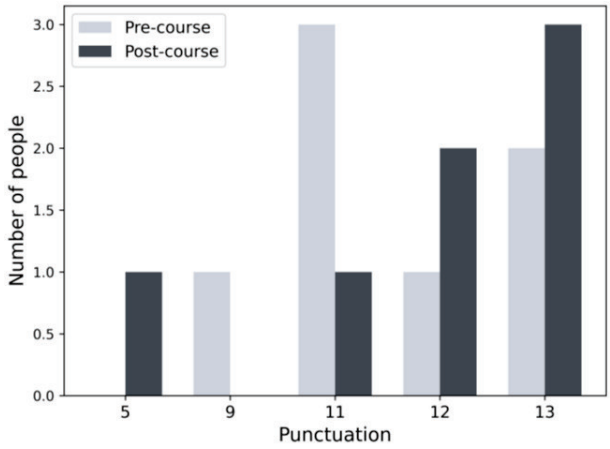

Figure 1. Comparison of pre and post Module II score distribution with Group A students.

When comparing the scores of this group, pre and post Module II, we observe that their variations are not as significant. This is because they already had access to information on the subjects covered in Module I of the project, using Scratch. This prior familiarity made it easier to answer the questionnaire and resulted in better performance compared to those who had no prior exposure to the subjects.

Results from Group B

In the analysis of Group B data, we observed a significant change between pre and post Module II assessments. In Figure 2, it can be seen that the number of correct answers increased from one assessment to the next, indicating a successful assimilation of the content covered during the project. It is noted that students who previously answered between five to nine questions correctly now answer between nine to twelve questions correctly, demonstrating a positive improvement in acquired knowledge.

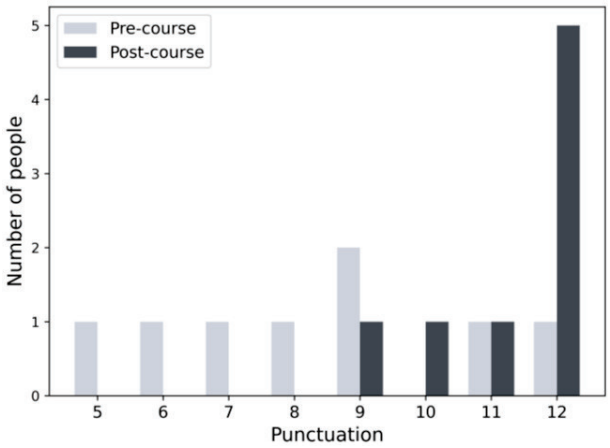

Figure 2. Comparison of the performance of Group B students before and after Module II.

In the survey, it was found that students who did not participate in Module I prior to Module II made a high number of errors on four specific questions: Q06, Q09, Q11, and Q13. The topics of these questions were covered in both modules, aiding in the learning and application for the students who participated in them.

When compared to the post-course evaluation, it is observed that these questions are no longer in that category; they show improved scores. Furthermore, there were no questions with over 50% errors, demonstrating the progress of students in this group and the impact of the course.

Comparison between Groups

In comparing the evaluations of Group A and Group B, a significant change in student development is noticeable. One can see the difference in initial levels between each group and how these levels adjust from one evaluation to the next. In the pre-course evaluation, Group B demonstrates greater difficulty in answering questions correctly, whereas Group A shows proficiency in resolving the questions.

It is observed that in questions Q09 and Q11, there were a higher number of errors from students due to the topic of conditional structure. Approximately 8 out of 15 students made errors on question Q09, and 9 out of 15 students made errors on question Q11. Among these students who made errors on the questions, only 3 for Q09 and 3 for Q11 had

been participants in Module I and had prior exposure to Scratch. The rest are beginners in Module II, without having gone through Module I before. The performance on the mentioned questions is shown in Figure 3.

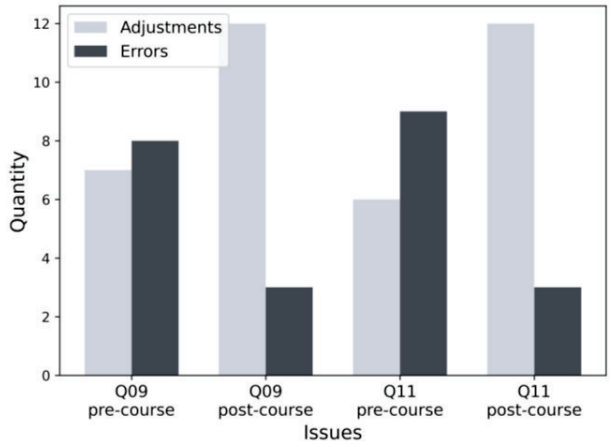

Figure 3. Comparison of questions Q09 and Q11 in the pre-course and post-course assessment of Module II.

In Figure 4, a clear distinction is observed between the range of 9 to 13 correct answers and the score of 5 correct questions. This range and the isolated score represent the number of questions students answered correctly and the number of students in each range. The graph depicted in the figure shows that 4 out of 15 students managed to answer the maximum number of questions correctly, while only 1 out of 15 students answered 5 questions correctly. Additionally, among the 4 students who achieved the maximum score, 3 had participated in Module I prior to Module II, as indicated.

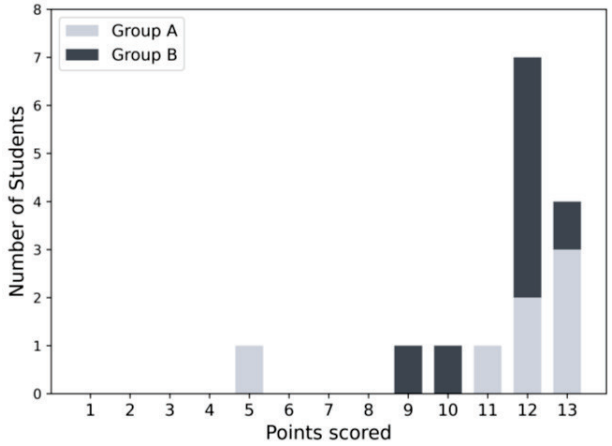

Figure 4. Distribution of students by number of correct answers in the post-Module II assessment.

Based on the results presented, it was possible to observe the relevance of the students' participation in Module I, as it establishes the necessary foundations to apply knowledge in a programming language. This allows students to develop a solid base that facilitates learning the language used in Module II, as there was a noticeable influence on the development of programming logic for problem-solving.

FINAL CONSIDERATIONS

This case study demonstrated that the use of Scratch, a block-based programming tool, can have a significant impact on the development of programming logic and computational thinking. The analysis of the two modules of the extension project "Iniciação à Programação," offered by IFPE - Campus Belo Jardim, revealed that introducing Scratch before teaching a traditional programming language like Python provided a more solid foundation for students, especially those with no prior experience. This initial approach significantly contributed to the enhancement of programming logic and computational thinking.

The results show that students who completed Module I with Scratch developed a more robust understanding of programming logic and computational thinking concepts, reflected in higher performance on subsequent Python assessments, compared to those who did not have this initial preparation. Scratch's visual and intuitive approach facilitated the understanding of programming fundamentals, preparing students more effectively for learning more complex textual languages.

Furthermore, this study reinforces the importance of democratizing access to programming knowledge, promoting digital inclusion, and reducing educational inequalities. Through more accessible and understandable teaching methodologies, such as the use of Scratch, it is possible to foster students' self-confidence in learning and applying new technological skills, better preparing them for the digital job market.

We conclude that using Scratch before introducing textual programming languages can be a valuable pedagogical strategy, promoting a more inclusive and technologically prepared environment. In addition, it is beneficial to use it before learning a programming language, to provide a solid foundation in fundamental concepts, such as programming logic and computational thinking. Future research could benefit from including more detailed statistical analyses, such as significance tests, to further deepen and reinforce the conclusions presented.

REFERENCES

1. Christian Brackmann. Brackmann's Computacional - Inicial. Retrieved from http://www.computacional.com.br/
2. Christ, N.A. 2019. Um estudo sobre a programação baseada em blocos. Trabalho de Conclusão de Curso, Fundação Educacional do Município de Assis, Assis.
3. Gobbi, R. 2020. Utilização do Scratch como ferramenta para apoiar o desenvolvimento do pensamento computacional. Trabalho de Conclusão de Curso, Universidade Federal de Santa Maria, Frederico Westphalen.

4. Nadya Intan Herawati, Heru Kuswanto, Mustika Wahyuni, and Anggriani Aristaria. 2024. Scratch-Assisted Computational Thinking in Physics: A Literature Review. JIPF (Jurnal Ilmu Pendidikan Fisika). Retrieved from https://doi.org/10.26737/jipf.v9i1.4696
5. Faíque Ribeiro Lima and Rogério Gomes. 2020. Conceitos e tecnologias da Indústria 4.0. Revista Brasileira de Inovação. Retrieved from https://doi.org/10.20396/rbi.v19i0.8658766
6. Vinicius Monteiro and Maristela Holanda. 2023. Pensamento Computacional e Scratch: Um relato de Experiências com Estudantes do Ensino Médio Público no Distrito Federal. In Anais do III Simpósio Brasileiro de Educação em Computação, abril 24, 2023, Evento Online, Brasil. SBC, Porto Alegre, Brasil, 254-261. Retrieved from https://doi.org/10.5753/educomp.2023.228348
7. No author found. 2018. Programação Em Blocos: Como Funciona? - I Do Code. Retrieved from https://idocode.com.br/blog/programacao/programacao-em-blocos/
8. No author found. Our Story — Scratch Foundation. Retrieved from https://www.scratchfoundation.org/our-story
9. Ricardo Rocha. 2019. Machine Learning e Programação Tradicional: as diferenças - Cegid Primavera. Retrieved from https://pt.primaverabss.com/pt/blog/machine-learning-e-programacao-tradicional/
10. A. Silva, Gustavo Araújo Brandão, P. Azevedo, and Deymes Aguiar. 2020. Usando a robótica educacional com Scratch e Arduino para melhor compreensão de Ciências Exatas. Scientia Prima. 6, 147–159. Retrieved from https://www.semanticscholar.org/paper/0148692b702f001755315b56733600c1d94b03e6
11. F. Silva, Alisandra Cavalcante Fernandes de Almeida, and K. A. D. Godoi e Silva. 2019. O DESENVOLVIMENTO DO PENSAMENTO COMPUTACIONAL COM A INTEGRAÇÃO DO SOFTWARE SCRATCH NO ENSINO SUPERIOR. Revista Observatório. 5(1), 276–298. Retrieved from https://doi.org/10.20873/UFT.2447-4266.2019V5N1P276
12. Alana Souza et al. 2023. Desenvolvendo o Pensamento Computacional Utilizando Scratch: Um Relato de Experiência da Formação de Professores da Educação Básica. In Anais do XXIX Workshop de Informática na Escola, novembro 06, 2023, Passo Fundo/RS, Brasil. SBC, Porto Alegre, Brasil, 918-929. Retrieved from https://doi.org/10.5753/wie.2023.234771
13. Souza, G. 2016. Linguagem de Programação Projetada para o Ensino da Programação em Português. Curso de Tecnologia em Análise e Desenvolvimento de Sistemas – Instituto Federal de Ciência e Tecnologia Farroupilha (IFFar) – Campus Alegrete, Alegrete.
14. Stewart, W. and Baek, K. 2023. Analyzing computational thinking studies in Scratch programming: A review of elementary education literature. International Journal of Computer Science Education in Schools. 6, 1 (Mar. 2023), 35–58. Retrieved from https://doi.org/10.21585/ijcses.v6i1.156
15. Simões, W.L., Pereira, L.M., Campello, L.H.P., Cavalheiro, R.A., e Ferreira, L.L.H. 2022. Habilidades profissionais no contexto da Indústria 4.0: um estudo da literatura. IV Simpósio de Engenharia de Produção, Universidade Federal de Catalão, Catalão.
16. Rhêmora Ferreira da Silva Urzêda, Eusiléa Pimenta Roquete Severiano, and Louis Dos Santos Amorim. 2020. O uso do scratch no curso de pedagogia: relato de uma experiência interdisciplinar. Workshop sobre Educação em Computação (WEI). Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 21-25. Retrieved from https://doi.org/10.5753/wei.2020.11122
17. L. M. Ventura, Luciane Guimarães Batistella Bianchini, and Lisandra Costa Pereira Kirnew. 2019. Scratch e a possibilidade de novos sentidos sobre o ensino da Lógica de Programação. Revista de Estudos e Pesquisas sobre Ensino Tecnológico (EDUCITEC). 5(11). Retrieved from https://doi.org/10.31417/educitec.v5i11.702
18. S. Vieira and Marcelo Sabbatini. 2020. CULTURA MAKER NA EDUCAÇÃO ATRAVÉS DO SCRATCH VISANDO O DESENVOLVIMENTO DO PENSAMENTO COMPUTACIONAL DOS ESTUDANTES DO 5° ANO DE UMA ESCOLA DO CAMPO DA CIDADE DE OLINDA-PE. Revista Docência e Cibercultura. 4, 43–66. Retrieved from https://doi.org/10.12957/redoc.2020.50671