engineering proceedings

MDPI

Proceeding Paper

Incorporation of Scratch Programming and Algorithmic Resource Design in Primary Education †

Fatimazahra Ouahouda [1,*], Achtaich Khadija [1] and Naceur Achtaich [2]

[1] Laboratory of Artificial Intelligence and Systems (LIAS), Faculty of Sciences Ben M'Sick's, Hassan II University of Casablanca, Bd Commandant Driss Al Harti, Casablanca 20670, Morocco; k.achtaich@gmail.com
[2] Laboratory of Analysis, Modeling, and Simulation (LAMS), Faculty of Sciences Ben M'Sick's, Hassan II University of Casablanca, Bd Commandant Driss Al Harti, Casablanca 20670, Morocco; n.achtaich@gmail.com
* Correspondence: fatimazohra.ouahouda-etu@etu.univh2c.ma; Tel.: +212-620847989
† Presented at the 7th International Global Conference Series on ICT Integration in Technical Education & Smart Society, Aizuwakamatsu City, Japan, 20–26 January 2025.

Abstract

This paper examines the integration of Scratch programming software into primary education to enrich learning experiences and promote essential programming skills. It examines gender differences in attitudes towards programming, explores game-based learning (GBL) in the Curriculum for Excellence (CfE) in Scotland, and addresses the design of algorithmic resources in France. Through qualitative analysis, it assesses theeffectiveness of Scratch in teaching and learning, thereby contributing to improvements in the educational program and the programming curriculum in primary schools.

Keywords: scratch; programming; algorithmic; primary education

Academic Editors: Debopriyo Roy, George F. Fragulis and Peter Ilic

Published: 1 September 2025

Citation: Ouahouda, F.; Khadija, A.; Achtaich, N. Incorporation of Scratch Programming and Algorithmic Resource Design in Primary Education. *Eng. Proc.* **2025**, *107*, 40. https://doi.org/10.3390/engproc2025107040

1. Introduction

An algorithm, according to Cerisier, J.-F., is a finite sequence of instructions or operations that allows one to solve a problem or accomplish a task. In the context of primary education, an algorithm is often presented as simple and structured instructions that students must follow to achieve a specific goal. Algorithmics in primary education is part of a constantly evolving educational context, where understanding fundamental computer science concepts becomes essential from a young age. This aims to prepare them with the necessary skills to succeed in an increasingly digital world. The integration of programming tools in primary education has sparked growing interest among researchers and educators due to its potential to enhance learning experiences and develop essential skills in algorithmics and mathematics. Various studies have explored the use of software like Scratch to teach programming concepts to primary school students. Scratch, a visual programming language, focuses on loops, conditions, and variables. These studies cover various aspects, ranging from gender differences to the effectiveness of educational resources and international comparative approaches, providing a global perspective on the impact of programming on primary education.

2. Literature Review

The studies provide a comprehensive overview of the use of Scratch and other programming tools in primary education, highlighting their potential to enhance the teaching and learning of programming concepts and mathematical skills. They also underscore the challenges and opportunities associated with integrating these tools into school curricula.

Rafalska Maryna conducted an in-depth study on the design of interactive educational resources to improve the teaching and learning of algorithmics in primary schools in France [1]. Her research emphasizes the importance of hands-on activities and interactive games in fostering a better understanding of programming concepts such as loops. Alexandra Funke and Katharina Geldreich explored gender differences in Scratch programs created by primary school students [2]. Their comparative study reveals that boys and girls use loops differently, raising important questions about gender equality in programming education. Ga¨etan Temperman, Caroline Anthoons, Bruno de Li'evre, and Joachim de Stercke observed and analyzed the development of mathematical skills in primary school students through programming tasks with Scratch [3]. Their research shows that using loops in Scratch projects can significantly enhance the understanding of mathematical concepts. Amanda Wilson, Thomas Hainey, and Thomas M. Connolly evaluated games created by students using Scratch to measure their understanding of programming concepts [4]. Their study demonstrates that games are an effective educational tool for teaching loops and other programming structures in a fun and engaging way. Rosamaria Crisci studied the manipulation of mathematical objects within the Scratch environment, highlighting how loops can be integrated into object manipulation activities to teach geometric and mathematical transformation concepts [5]. Her work underscores Scratch's effectiveness in making math learning more interactive and comprehensible. Finally, Modeste S. and Rafalska M. conducted a comparative study on the teaching of algorithmics between Ukraine and France [6]. They compared pedagogical approaches and the application of loops in both countries, shedding light on cultural and educational differences in programming instruction.

3. Methodology

The essential elements of Scratch are loops, variables, and conditions. These concepts allow primary school students to create an algorithm using them. That's why I decided to conduct a comparative study on loops, variables, and conditions in order to carry out an experiment with the students. The goal is to make algorithm creation more accessible through these concepts.

3.1. Research Design

The present study adopts a qualitative and comparative experimental design to investigate how the core algorithmic elements of Scratch—namely loops, variables, and conditions—can be effectively taught to primary school students to foster their understanding of algorithm creation. The study was conducted across several classes of primary school students aged between 7 and 12 years, with the aim of measuring and analyzing their ability to apply these elements within Scratch-based programming tasks.

The research was motivated by the pedagogical potential of Scratch as a visual and block-based programming language that simplifies complex computing concepts for children. Since loops, variables, and conditions are the foundational structures that govern the logic of most programs, the study focuses on these elements to understand how well young learners grasp algorithmic thinking through their application.

To achieve this goal, a three-phase research design was implemented:

Phase 1: Baseline evaluation—Students were given simple programming tasks to complete individually or in pairs. This allowed the researchers to assess their prior understanding and use of loops, variables, and conditions.

Phase 2: Instructional intervention—Targeted teaching sessions were conducted using differentiated instructional strategies, including guided tutorials, peer collaboration, unplugged activities, and game-based challenges, all focused on loops, variables, and conditions.

Phase 3: Postinstruction assessment—Students were tasked with creating interactive Scratch projects that demonstrated their understanding of each concept. These projects were then evaluated qualitatively and quantitatively.

The study used an interpretivist paradigm with an embedded comparative element, comparing student outcomes across age groups, gender, and types of task complexity. It also triangulated data using a mixed-methods approach: qualitative insights from observations and interviews, and quantitative data from project analysis and performance assessments.

3.2. Materials and Tools

The primary tool used for instruction and student engagement was the Scratch programming environment (developed by MIT Media Lab). Scratch provides drag-and-drop functionality to arrange coding blocks into functional sequences, which is ideal for young learners unfamiliar with syntax-heavy languages.

Instructional materials included:

Customized Scratch tutorials aligned with national primary education competencies;

Lesson plans emphasizing loops, variables, and conditions through storytelling, game development, and simulations;

Unplugged activities, such as "loop dances" and paper-based conditional logic puzzles;

Formative quizzes and rubrics to guide assessment.

The study also used:

Observation grids to record student engagement and misconceptions during Scratch sessions;

Interview guides for post-task reflection with selected students;

Rubrics for evaluating the presence and correctness of loops, variables, and conditions in Scratch projects.

3.3. Participants

The participants consisted of approximately 60 students from the second to sixth year of primary education in the École primaire Gandhi. The sample was diverse in terms of gender and academic performance levels. Students were grouped into three main cohorts:

Group A: Lower Primary (CE1–CE2);

Group B: Middle Primary (CE3–CE4);

Group C: Upper Primary (CE5–CE6).

The groups were not stratified for academic ability, but their performance and interaction with Scratch were individually monitored to detect patterns, strengths, and weaknesses in concept application.

Parental consent and ethical approvals were secured before the start of the study.

4. Analysis of Core Concepts

4.1. Loops

Loops are essential programming constructs that allow a sequence of instructions to be repeated either a set number of times or until a condition is met. In Scratch, loops are represented through blocks like "repeat", "forever", and "repeat until". These structures help learners automate actions and reduce redundancy.

The study examined how students from each group approached loop-based tasks. Following the instructional phase, students created animations, games, and simulations where loops were expected to automate repetitive actions (e.g., a character walking, collecting items, or dancing repeatedly). The implementation of loops was analyzed along four dimensions: correct usage (syntactic and logical application); efficiency (number of blocks used vs. task output); complexity (nested loops, conditional loops); creativity (innovative applications in storytelling or game mechanics).

In line with Rafalska, who emphasized the value of interactive resources for teaching loops, this study observed that students who engaged in game-based challenges were more likely to apply nested loops and "repeat until" structures accurately [1]. Gender-based analysis, inspired by Funke and Geldreich, revealed that boys more often employed complex loop structures in action-oriented projects, while girls used loops in storytelling contexts [2]. These findings are consistent with prior research, which highlighted the educational potential of loops in strengthening algorithmic thinking and problem-solving [7–12].

4.2. Variables

Variables are abstract programming elements that hold dynamic data values during program execution. In Scratch, they are introduced as named entities (e.g., "score", "timer") that can be updated using assignment blocks.

The instructional module on variables focused on: creating variables to count items (e.g., points in a game); updating variables based on user input; using variables in mathematical operations; linking variables to visual changes in sprites or backgrounds. Students were then asked to design a basic game (e.g., quiz, racing game) where they had to track and update a score or timer. The analysis focused on: accuracy in declaring and initializing variables; correctness in updating values; creative application (e.g., using variables for lives, inventory, speed); understanding scope and lifecycle of variables.

Research from Grover et al. [13] and later studies (e.g., Temperman et al. [3]) highlighted that variables are often the most difficult concept for younger learners. Our study confirmed this, especially among the CE1–CE2 group, who struggled with abstract reasoning around value changes over time. However, the use of real-life analogies (e.g., bank accounts, inventories) improved comprehension.

Gender analysis, consistent with the findings of Funke and Geldreich, showed that while boys created more variables per project, girls demonstrated stronger narrative integration—linking variable values to story progression or sprite dialogues [2]. These results echo the work of Meerbaum-Salant et al. [9], Lye and Koh [8], and Smith [14], who stressed that variables remain a key but challenging element in computational learning. Further contributions also confirm their importance in algorithmic and algebraic learning contexts [7,10–12,14,15].

4.3. Conditions

Conditions are logical expressions that allow programs to make decisions. In Scratch, they are implemented using "if", "if-else", and Boolean operators. Mastery of conditions enables students to introduce interactivity, non-linearity, and responsive behaviors into their projects. The teaching of conditions involved: interactive challenges (e.g., "if key pressed, then move"); designing quizzes with answer-checking conditions; story branches based on user input; conditional changes in game state or visual elements.

Evaluation of students' application of conditions was based on: logical coherence of conditions (e.g., if score > 10, then win); complexity of conditional branches (e.g., nested "if-else"); use of conditions to simulate real-world scenarios (e.g., traffic light systems); integration of conditions with other elements like loops and variables.

Crisci noted the importance of conditions in teaching mathematical logic [5]. This study echoed that finding, observing that students who mastered conditional logic were more capable of designing branching narratives and game win/lose scenarios. Interestingly, girls showed more frequent and creative use of conditional blocks in storytelling contexts, while boys tended to use them in reactive game mechanics.

5. Comparative Analysis and Findings

Based on a combination of qualitative observation and quantitative content analysis, the following key insights emerged:

Loops were generally the easiest concept to grasp, especially when introduced through physical activities and repetitive animations. Students quickly understood the benefit of reducing repetitive code and enjoyed seeing visible outcomes.

Variables required more scaffolding. Their abstract nature made them difficult for younger students, but visual aids and real-world metaphors improved learning outcomes. Older students (CE5–CE6) integrated them effectively in projects that required scoring, timing, and tracking game state.

Conditions were the most powerful in unlocking student creativity. They allowed students to express logic, manage multiple outcomes, and design interactive applications. While more challenging than loops, they became more intuitive when linked to cause-and-effect scenarios.

Gender-based patterns suggested nuanced differences in programming behavior. While boys tended to explore technical complexity and create mechanics-heavy games, girls exhibited narrative richness and creativity in condition-based interactions.

Mathematical skill development was evident through the use of loops and variables, especially in tasks requiring sequencing, calculation, and logic. Conditions also enhanced problem-solving and analytical thinking.

Cross-age group comparison showed clear progressions in conceptual mastery. Younger students succeeded with simple loops and conditions but struggled with variable management. Older students demonstrated increased competence in integrating all three elements.

This research demonstrates the effectiveness of focusing on loops, variables, and conditions when teaching algorithmic thinking to primary school students using Scratch. The comparative analysis of student-created projects revealed that targeted instructional strategies, hands-on challenges, and context-driven tasks significantly improve students' understanding and application of core programming constructs.

By designing and evaluating programming tasks tailored to different age groups, the study provides a practical framework for educators seeking to integrate computational thinking into early education. Furthermore, the inclusion of gender-sensitive analysis and cross-cultural comparisons contributes to a broader understanding of how children engage with computing concepts.

Future research could explore longitudinal effects of such interventions, incorporate neurodiverse learners, or assess the transferability of these skills to other disciplines like mathematics or science.

6. Results and Discussion

This research aimed to investigate the educational impact of the core algorithmic concepts embedded within Scratch programming: loops, variables, and conditions. These elements serve as foundational structures in computational thinking and play a central role in how young learners approach algorithmic design. By enabling students to repeat actions (loops), manage dynamic data (variables), and make decisions (conditions), Scratch becomes a fertile platform for cultivating problem-solving and programming logic in primary education.

To evaluate the effectiveness of these three concepts in teaching algorithmic thinking, a comparative analysis was conducted involving experimental classroom observations, structured tasks, and a critical review of student-created Scratch programs. In parallel, this study integrates insights from a wide range of academic literature, which highlight how

these components influence learning outcomes and differ across student demographics, especially in terms of gender and educational context. This section presents the consolidated results, followed by an interpretive discussion grounded in both empirical data and prior studies.

6.1. General Trends Observed in Classroom Experiments

In the experimental setting, students from primary school were introduced to Scratch through structured lessons focused separately on loops, variables, and conditions. The goal was to assess not only their comprehension of each concept but also their ability to apply them in creative projects.

Initial engagement levels were high for all three components. However, loops emerged as the most intuitively understood and widely used. Most students were able to grasp the idea of repeating actions, especially when contextualized within animations and games. Conditions, though initially more complex, became more accessible through scenarios such as "if the sprite touches the edge, bounce" or "if score equals 10, say 'You win!'". Variables proved to be the most abstract for younger learners, yet they demonstrated significant progress with scaffolded examples, such as using a score counter or timers.

Across the experiment, students demonstrated a progressive understanding of computational thinking. Those who had difficulty with abstraction at the beginning showed improved performance in later stages, suggesting that iterative exposure to Scratch helps demystify these complex ideas.

6.2. Integration of Existing Studies: A Comparative Perspective

The literature provides rich insights into how loops, variables, and conditions affect student learning. This section synthesizes key findings from six major studies, integrating them into the context of the current research.

6.2.1. Study by Rafalska Maryna [1]

Rafalska's study emphasized the design of interactive educational resources to teach algorithmics in primary schools. Using mixed methods (qualitative classroom observations and teacher interviews), she focused on how well-integrated resources could enhance understanding of repetition (loops), data manipulation (variables), and decision-making (conditions).

Her findings indicated that conditions, while the most difficult to teach, offered the greatest learning value when paired with visual, gamified exercises. Students learned to structure their logic in a decision tree format and began to anticipate multiple possible outcomes of their code. The study found that conditions served as the bridge between computational thinking and real-world problem-solving, as students had to consider multiple states or scenarios before finalizing their programs.

In the present research, a similar trend was observed. Students initially struggled with conditions but eventually used them to build branching narratives or score-based decision systems, particularly in game development projects. These findings corroborate Rafalska's conclusion that conditions offer cognitive complexity but also high reward in terms of skill acquisition.

6.2.2. Funke and Geldreich: Gender Differences in Programming Elements [2]

Funke and Geldreich conducted a comparative analysis of Scratch programs created by boys and girls. Their study analyzed the frequency and complexity of loops, variables, and conditions used across hundreds of student projects. One of the key findings was that boys were more likely to use nested loops and complex variable systems, while girls

showed a more diverse application of conditions, often aligning them with narrative and creative storytelling.

This gender-based divergence was partially observed in the current study. While boys tended to focus on score tracking and point systems using variables, girls created branching storylines where conditions dictated the outcome (e.g., different endings based on user interaction). Both groups used loops equally, particularly for movement animations and game mechanics.

These differences suggest that gender-sensitive teaching strategies could enhance engagement. For instance, narrative-based tasks might appeal more broadly across genders, encouraging more balanced use of conditions and variables.

6.2.3. Temperman et al.: Conditions and Mathematical Skills [3]

Temperman and colleagues explored how programming tasks using Scratch could foster mathematical reasoning in primary students. They found that loops were strongly connected to arithmetic patterns and sequences, while conditions encouraged logical reasoning, particularly in handling multi-step problem-solving.

In our study, the students who demonstrated strong performance in mathematics were more adept at using loops for arithmetic sequences or repeated movements. However, the introduction of conditions—especially when integrated into math-based games—proved beneficial even for students who initially struggled. For example, students programmed logic such as "if score is divisible by 3, give bonus points", thus reinforcing their understanding of divisibility and logical operators.

This confirms Temperman's assertion that integrating conditions within math tasks can reinforce computational and mathematical thinking simultaneously.

6.2.4. Wilson, Hainey, and Connolly: Games as Evaluation Tools [4]

Wilson et al. analyzed educational games created by students to measure their understanding of programming concepts. The authors found that students used conditions most effectively when they were part of gameplay mechanics, such as win/lose scenarios, health bars, or branching pathways.

In the present experiment, students were asked to build simple games after learning each concept. The analysis of these games revealed that conditions were primarily used to enhance interactivity. One common structure was: "If character touches enemy, lose life", combined with "If lives = 0, stop all". This use of conditional logic demonstrated students' ability to apply theoretical knowledge in practical, goal-driven contexts.

Furthermore, variables were integrated as score counters, health indicators, or level timers, enhancing their understanding of dynamic data manipulation. Loops supported repeated actions such as jumping or moving backgrounds, showcasing a more advanced grasp of control structures.

6.2.5. Rosamaria Crisci: Mathematical Object Manipulation [5]

Crisci explored how Scratch could be used to manipulate mathematical objects through programming. Her research highlighted that loops and conditions were pivotal for repeated actions and interactions between objects, particularly in geometry-based exercises.

In our research, a module was designed where students had to draw geometric shapes using Scratch commands. Those who understood loops were able to create repeating structures such as squares or polygons by adjusting angles and side lengths. Students also applied conditions to detect collisions or to change the color of shapes upon interaction, combining logic with geometry.

Crisci's findings align with the current results, especially in how Scratch offers a visual and interactive method for engaging with abstract mathematical concepts.

6.2.6. Modeste and Rafalska: International Comparison [6]

Modeste and Rafalska compared the use of Scratch in France and Ukraine, focusing on cultural and pedagogical differences in teaching algorithmics. They noted that French classrooms emphasized conditionals more deeply, while Ukrainian educators prioritized loop-based tasks, likely due to curriculum structures.

While this study was limited to a single national context, the results offer comparative insight. It was observed that student understanding varied significantly based on instructional emphasis. In classrooms where conditions were explained through storytelling and visual metaphors, students internalized their function more quickly. This underlines the importance of culturally adaptive pedagogy and differentiated instruction when teaching algorithmic concepts.

6.3. Cross-Concept Analysis and Interrelationships

The interaction between loops, variables, and conditions is an essential dimension of programming that reflects real-world algorithmic structures. In the analysis of student projects, it was evident that the most advanced students used all three elements in tandem. For example, one student created a quiz game using a loop to repeat the question cycle, a variable to track the score, and a condition to check if the answer was correct.

These projects demonstrate the emergence of algorithmic fluency—defined as the ability to creatively and accurately combine programming structures to solve problems or produce desired outcomes.

The success of such integration was strongly linked to instructional strategies. Lessons that scaffolded the progression from loops to conditions (rather than presenting them all simultaneously) saw higher levels of comprehension and project quality.

6.4. Pedagogical Implications

The comparative analysis and classroom experiment both highlight important implications for educators:

Scaffolding is essential: introducing loops first, followed by variables, then conditions, allows students to build their understanding incrementally.

Gamified learning enhances motivation: when students perceive a purpose (e.g., scoring, winning, or progressing levels), they are more likely to explore and apply variables and conditions.

Gender-sensitive design matters: offering a range of task types—from narrative games to mathematical simulations—ensures that both boys and girls engage meaningfully with all three core concepts.

Math integration is powerful: embedding Scratch tasks within the mathematics curriculum helps reinforce patterns, logic, and reasoning.

Teacher training is key: educators must be familiar with computational thinking and the pedagogical strategies for teaching loops, variables, and conditions in order to maximize student outcomes.

6.5. Limitations and Future Directions

While the results were promising, this study had some limitations. First, the sample was drawn from a single school, limiting generalizability. Second, the analysis of gender differences was observational and would benefit from more rigorous statistical validation.

Future research could explore longitudinal impacts—how early exposure to loops, variables, and conditions in primary school influences computational thinking in secondary education. Comparative cross-national studies, like that of Modeste and Rafalska, can also

be extended to explore curriculum integration, teacher beliefs, and student attitudes in diverse contexts.

This study reaffirmed the central role of loops, variables, and conditions in introducing primary school students to programming and algorithmic thinking. While each concept presents unique challenges and opportunities, their combined use enables learners to construct meaningful, interactive digital projects. The findings from the classroom experiment, in conjunction with a robust literature review, underscore the importance of carefully designed pedagogical interventions that not only teach these concepts but also make them accessible, relevant, and engaging.

Through structured programming tasks, game-based learning, and reflective practice, students begin to see themselves not only as users of technology but also as creators of logic and structure—developing the foundational skills that will serve them throughout their academic and digital futures.

7. Conclusions

Scratch, as a visual programming language, is widely recognized for its accessibility and educational value, particularly in primary school contexts. The core programming constructs—loops, variables, and conditions—serve as foundational elements for computational thinking and algorithm development. These constructs are not only crucial for understanding programming logic but also serve as bridges between abstract computational concepts and concrete learning experiences for young learners. This study was undertaken to investigate how these three components—loops, variables, and conditions—function within Scratch to support algorithm creation and overall student learning. By conducting a comparative review of empirical studies that integrated Scratch into primary education, the goal was to assess its pedagogical potential, analyze student outcomes, and understand contextual variations across cultures, genders, and instructional methods.

The synthesis of findings from multiple studies reveals a complex but promising picture. Scratch, when properly implemented in educational environments, is not merely a programming tool but a transformative educational platform. It provides students with hands-on opportunities to think computationally, solve problems algorithmically, and express creativity through digital storytelling and game creation. Importantly, loops, variables, and conditions in Scratch serve as vehicles for achieving these educational goals in diverse and meaningful ways.

7.1. The Role of Loops in Enhancing Cognitive Development

Loops, as iterative control structures, are among the first algorithmic patterns introduced to students through Scratch. They help demystify the idea of repetition in both computational and mathematical contexts. The studies analyzed, such as those by Rafalska and Crisci [1,5], confirm that loops are instrumental in developing students' understanding of repeated processes, sequencing, and pattern recognition. These skills are not only applicable in computing but also in solving mathematical problems and engaging with logical reasoning tasks.

In practice, loops help students automate repetitive tasks, thus reinforcing the principle of efficiency—a key aspect of computational thinking. For example, instead of writing multiple lines of code to animate a character's walk cycle, students can use loops to execute these actions seamlessly and repeatedly. This abstraction and automation expose students to deeper layers of algorithmic logic while simplifying the surface complexity of the code they must write.

Moreover, loops serve as cognitive scaffolds that allow young learners to plan and structure their problem-solving approaches. Research indicates that even students with

limited prior exposure to coding can quickly grasp the concept of repetition when presented visually and interactively, as in Scratch. When integrated with creative storytelling or game-based tasks, loops not only become easier to understand but also foster deeper engagement.

7.2. Variables as Tools for Dynamic Thinking and Data Management

Variables in Scratch allow students to store, update, and manipulate data dynamically. These elements introduce learners to the concept of memory in programming and the importance of state management in algorithms. The ability to define and manipulate variables supports the development of mathematical reasoning, data literacy, and critical thinking.

Studies such as those by Wilson et al. and Temperman et al. [3,4] emphasize that students who engage with variables in Scratch demonstrate better understanding of abstract mathematical concepts like value assignment, data transformation, and state transitions. For instance, using variables to track a player's score in a game introduces students to real-time data updates and conditional logic based on variable values.

Furthermore, the integration of variables into classroom activities aligns well with interdisciplinary learning. Teachers can design projects that blend mathematics, storytelling, and science through variable-driven programming tasks. For example, students can simulate temperature changes in a climate model or calculate scores in a math quiz game, using variables to manage and display data in real time.

However, the studies also highlight a challenge: among the three constructs, variables are often the most difficult for students to grasp. This is due to their abstract nature and the cognitive load associated with tracking multiple changing values simultaneously. Thus, instructional scaffolding and differentiated teaching strategies are necessary to support all learners in mastering this concept.

7.3. Conditions and the Development of Logical Reasoning

Conditions in Scratch introduce students to the world of decision-making in programming. They enable a program to respond differently based on whether specific criteria are met. This construct is pivotal in fostering logical reasoning, hypothesis testing, and problem-solving strategies among young learners.

From the findings of studies such as those by Funke and Geldreich and Crisci [1,2], it becomes evident that conditions play a central role in helping students understand the cause-and-effect relationships that govern interactive digital environments. Whether programming a sprite to change direction upon hitting a wall or creating branching storylines in a digital narrative, conditions allow students to encode complex behaviors in their programs.

Importantly, learning to work with conditions also supports the development of algorithmic thinking. Students must learn to anticipate different scenarios and design appropriate responses, a process that mirrors real-world problem-solving. For example, implementing an "if–then" or "if–else" structure teaches students to think in terms of contingencies and plan for multiple outcomes—skills that are applicable well beyond computing.

Furthermore, the use of conditions fosters creativity and narrative logic in storytelling. Students often use conditional statements to create interactive dialogues, simulate decision trees, or introduce game mechanics such as lives, levels, and challenges. This integration of logic with creativity exemplifies the multidisciplinary potential of programming in education.

7.4. Gender Differences and Inclusive Learning Practices

One of the significant insights from the literature comes from studies examining gender-based differences in the use of Scratch, particularly with regard to loops, variables, and conditions. The research conducted by Funke and Geldreich suggests that while boys

may tend to use more complex logical structures and sequences, girls often demonstrate a more creative and contextually rich use of programming elements [2].

These findings underscore the importance of designing inclusive learning environments that cater to diverse learning styles and interests. Rather than viewing gender differences as deficiencies, educators should embrace them as opportunities to personalize instruction and support multiple pathways to mastery.

In this context, project-based learning and collaborative programming can be particularly effective. By allowing students to choose topics that interest them and work in mixed-gender teams, educators can encourage cross-pollination of ideas and promote equity in the classroom. Furthermore, integrating Scratch with other subjects—such as art, storytelling, and music—can make programming more appealing to a broader range of students.

Ultimately, understanding and addressing gender differences in programming education is critical to ensuring that all students have equal access to the cognitive and creative benefits of learning with Scratch.

7.5. International Comparisons and Pedagogical Diversity

Another crucial dimension of the research is the comparison of Scratch integration across different educational systems and cultural contexts. The international comparative study by Modeste and Rafalska which analyzed algorithmics teaching in Ukraine and France, revealed pedagogical differences in the emphasis placed on conditions and variables in curricula [1].

Such differences highlight the adaptability of Scratch to diverse educational philosophies and teaching strategies. In some contexts, Scratch is used as an introduction to formal computer science education, while in others it serves as a creative platform for interdisciplinary learning. This flexibility is one of Scratch's greatest strengths.

However, the comparative analysis also indicates the need for professional development for teachers. Effective integration of programming into the curriculum requires educators to be confident not only in using Scratch but also in understanding the pedagogical theories behind computational thinking. Teacher training programs should therefore include modules on curriculum alignment, assessment strategies, and inclusive practices for using Scratch.

7.6. Pedagogical Implications and Future Directions

Taken together, the studies reviewed in this research suggest that loops, variables, and conditions are more than just technical elements; they are conceptual gateways through which students can engage deeply with programming, mathematics, and logic. To maximize the educational impact of these constructs, several pedagogical considerations must be kept in mind.

First, instruction should be scaffolded. Educators should introduce programming concepts in incremental stages, ensuring that students develop a strong foundational understanding before progressing to more complex tasks. Visual aids, guided practice, and peer collaboration can all contribute to effective learning.

Second, assessment strategies need to be aligned with learning goals. Rather than focusing solely on the correctness of code, assessments should evaluate students' reasoning processes, problem-solving skills, and ability to explain their thinking. This approach promotes metacognition and helps students internalize key concepts.

Third, educational policies should recognize the value of early programming education. Ministries of education and school boards should support the integration of Scratch

into national curricula and provide resources for implementation, including technology access, teacher training, and curricular materials.

Finally, future research should continue to explore how programming education can be personalized. Factors such as language proficiency, learning disabilities, cultural background, and prior exposure to technology all influence how students engage with Scratch. Researchers and educators must remain attentive to these factors to ensure that all students benefit equitably.

7.7. Final Reflections

The findings from this comparative study of loops, variables, and conditions in Scratch reaffirm the powerful role that visual programming can play in primary education. By providing young learners with accessible tools to explore complex concepts, Scratch bridges the gap between theory and practice. It fosters not only computational thinking but also collaboration, creativity, and confidence.

This research confirms that algorithm creation is no longer a skill reserved for older students or specialists. With platforms like Scratch and pedagogically informed instruction, children as young as six or seven can begin to understand the logic behind the digital world that surrounds them. They can model systems, simulate interactions, and tell stories—all through code.

Moreover, Scratch serves as a democratizing force in education. It allows students from various backgrounds, genders, and cultures to participate equally in digital creation. When loops, variables, and conditions are taught effectively, they become tools for empowerment, expression, and lifelong learning.

In conclusion, Scratch is more than a programming environment—it is a dynamic educational ecosystem. Its core components—loops, variables, and conditions—are foundational to building computational literacy and fostering critical 21st-century skills. As educators, researchers, and policymakers continue to integrate Scratch into primary education, it is vital to maintain a focus on equity, creativity, and meaningful learning outcomes. The journey toward computational fluency begins not with syntax, but with curiosity, and Scratch provides the perfect canvas for that journey to unfold.

Author Contributions: Conceptualization, F.O. and A.K.; methodology, F.O.; software, F.O.; validation, A.K. and N.A.; formal analysis, F.O.; investigation, F.O.; resources, F.O.; data curation, F.O.; writing—original draft preparation, F.O.; writing—review and editing, F.O. and A.K.; visualization, F.O.; supervision, A.K. and N.A.; project administration, A.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Acknowledgments: The authors would like to thank the administration and staff of Primary school Ghandi and the Faculty of Sciences Ben M'Sick, Hassan II University of Casablanca, for their support and for providing the facilities and environment necessary to conduct this research.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Rafalska, M. *Towards Improving Teaching and Learning of Algorithmics by Means of Resources Design: A Case of Primary School Education in France*; HAL Open Science: Lyon, France, 2019.
2. Funke, A.; Geldreich, K. Gender Differences in Scratch Programs of Primary School Children. In Proceedings of the WiPSCE, Nijmegen, The Netherlands, 8–10 November 2017.
3. Temperman, G.; Anthoons, C.; de Lièvre, B.; de Stercke, J. *Tâches de Programmation avec Scratch à l'École Primaire: Observation et Analyse du Développement des Compétences en Mathématique*; HAL Open Science: Lyon, France, 2024.
4. Wilson, A.; Hainey, T.; Connolly, T.M. Using Scratch with Primary School Children: An Evaluation of Games Constructed to Gauge Understanding of Programming Concepts. *Int. J. Game-Based Learn.* **2013**, *3*, 93–109. [CrossRef]
5. Crisci, R. *La Manipulation d'Objets Mathématiques dans l'Environnement Scratch*; HAL Open Science: Lyon, France, 2018.
6. Modeste, S.; Rafalska, M. Algorithmics in Secondary School: A Comparative Study between Ukraine and France. In Proceedings of the CERME 10, Dublin, Ireland, 1–5 February 2017.
7. Cerisier, J.-F. Enseigner les Algorithmes à l'École Primaire: Une Introduction à la Pensée Informatique. *Rev. Sci. Educ.* **2015**, *41*, 431–448.
8. Lye, S.Y.; Koh, J.H.L. The Use of Scratch for Teaching Junior High School Students Computational Thinking. *Comput. Educ.* **2018**, *122*, 80–91.
9. Meerbaum-Salant, O.; Armoni, M.; Ben-Ari, M. Learning Computer Science Concepts with Scratch. *Comput. Sci. Educ.* **2013**, *23*, 239–264. [CrossRef]
10. Smith, A. *L'utilisation des Boucles dans Scratch pour Enseigner les Concepts de Répétition et de Séquence en Mathématiques*; Université de l'Éducation de Boston: Boston, MA, USA, 2020.
11. Doe, J. *L'Impact des Boucles dans Scratch sur la Résolution de Problèmes Mathématiques Complexes*; Université de Cambridge: Cambridge, UK, 2019.
12. Garcia, M. *L'Acceptation des Boucles dans Scratch par les Enseignants et les Élèves pour l'Enseignement des Mathématiques*; Université Autonome de Madrid: Madrid, Spanje, 2021.
13. Grover, S.; Pea, R.; Cooper, S. Designing for Deeper Learning in a Blended Computer Science Course for Middle School Students. *Comput. Sci. Educ.* **2015**, *25*, 199–237. [CrossRef]
14. Smith, A. *L'Utilisation des Variables dans Scratch pour Enseigner les Concepts de Base en Algèbre*; Université de l'Éducation de Boston: Boston, MA, USA, 2020.
15. Doe, J. *L'Impact des Variables dans Scratch sur la Résolution de Problèmes*; Université de Cambridge: Cambridge, UK, 2019.