

Article

JLcoding Language Tool for Early Programming Learning

Wei-Ying Li and Tzu-Chuen Lu * 

Department of Information Management, Chaoyang University of Technology, Taichung 41349, Taiwan; totoro@y-shun.com.tw

* Correspondence: tclu@cyut.edu.tw

Abstract: This paper proposes a symmetry language of block-based to design novel educational programming called the JLcoding system. JLcoding system helps students convert from a block-based language to a text-based programming language. The interface and function of the system are block-based programs such as Scratch, but it is designed with text-based architecture. The system contains graphic teaching to teach the basic knowledge of programming, such that students can maintain interest and confidence when learning computational thinking. The system simultaneously combines the advantages of block-based and text-based programming. This research engaged 41 students who learned block-based programming language as the research objects. The experimental results show that the students can obtain higher post-test scores than the pre-test scores after learning the JLcoding system. The degree of learning progress was not affected by their gender. Additionally, it was discovered that male students have higher confidence in their programming abilities, and students who have learning interests are more motivated to continue learning the program.

Keywords: text-based programming; block-based programming; JLcoding; early programming learning



Citation: Li, W.-Y.; Lu, T.-C. JLcoding Language Tool for Early Programming Learning. *Symmetry* **2022**, *14*, 1405. <https://doi.org/10.3390/sym14071405>

Academic Editors: Alexander Shelupanov and Sergei D. Odintsov

Received: 25 May 2022

Accepted: 4 July 2022

Published: 8 July 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

1.1. Background

Currently, many secondary schools have included computer science in their school curricula [1–7]. Schools conduct this practice for several reasons. First, to prepare students to study and advance their careers in computer learning [8,9]. Second, there is recognition that the skills to express computational-based ideas play an important role for students to be able to take part in the digital age [10]. Third, computer programming is a major digital literacy competency of the twenty-first century, and the learning process of programming is thought to develop computational thinking. The idea of computational thinking was introduced by Jeannette Wing in a short article entitled “Computational Thinking”. Computational thinking involves problem-solving, system design, understanding human behavior, and delineating basic computer science concepts. According to Jeannette Wing, the ability to think computationally will make students better at completing their daily tasks. She also suggested that computational thinking should be integrated into subjects in schools [11,12]. Finally, computer programming is considered to provide opportunities for students to develop their intellectual abilities in overcoming challenging problems in the digital age [13]. Besides computational thinking, creativity, critical thinking, and problem-solving are important skills for students in the twenty-first century. These practices aim to make students take part in an active and participatory manner as knowledge creators rather than just as passive consumers of information [14]. Students in the past were considered only recipients of culture and knowledge. Apparently, in the current digital age, students must be actively involved in social meaning-making activities [15]. The current situation requires humans with deep knowledge, practical imagination, creative participation, intellectual curiosity, and collaborative commitment [16].

Programming, however, is not easy to learn at both the elementary, middle, and college levels [17–20]. Beginner programmers often make many mistakes writing the programs [21]. The most challenging task for students in primary and secondary schools is in understanding the fundamentals of programming, which will enable them to develop algorithmic and computational thinking in programming [22–24], preventing them from becoming stuck in negative attitudes toward programming [25].

Therefore, many school curricula choose to use block-based programming or programming languages designed for educational purposes to ease the burden on students to learn to program, such as Ozoblockly, Blockly, Snap, Scratch, Alice, Code Spells, or Lego Mindstorms [1,9]. These programming languages make it easy for students to learn programming by simply dragging and dropping their code blocks to form valid and meaningful instructions to achieve the desired results [26]. The advantage of this method is that the blocks are similar to jigsaw puzzles or Lego blocks that can be assembled specially to create a program for students who can easily create simple programs to boost their motivation.

1.2. Motivation

Students' perceptions of programming come with difficulties; however, educational programming can be more accessible, less frustrating, and beneficial for young students, but it might not achieve the objective of building students' interest and confidence in the field of computer programming. A student can perform well in computer science but not feel sure of what he/she is learning [9]. The attention of the researchers then focused on educational programming that was more effective in encouraging computational thinking [26].

Several researchers have researched and compared the educational programming that is most widely used in the school curriculum. The researchers [13] compared Logo and Scratch, and their findings show that students who used Scratch performed better on the programmer's competency assessment, but they lacked confidence in their programming skills; however, students who used Logo had more positive self-confidence. These findings show that students who used Scratch may not be aware that they are learning computer programming [9]. The researchers [26] compared Scratch with the App Inventor. Their findings show that there are clear differences between the two programming languages. Students using Scratch scored higher on average in Parallelism, Synchronization, and Flow Control, while students using App Inventor scored higher average scores on User Interactivity and Data Representation. In addition, they also found that the overall scores increased the size and type (genre) of the program. The researchers [27] compared the interfaces of block-based and text-based programming in introductory programming. Their findings show that a block-based programming interface can improve the performance of introductory programming, especially in terms of the speed of attaining programming goals. Nevertheless, to the question of whether there was an influence on students' perceptions of programming difficulties, their findings did not provide enough survey evidence to support this claim; however, there was little correlation between perceived difficulty and goal completion. Similar research was also conducted by the researchers [10]. They compared block-based programming with text-based programming for the same programming environment. Their findings show that students who used block-based programming showed greater learning outcomes and higher levels of interest, while students who used text-based programming saw their programming experience as more comparable to professional programmers and more effective at improving their programming skills; the researchers did not find any significant differences between the two groups of students for problems of self-confidence and pleasure.

Nevertheless, apart from the various studies stated, students still experience great difficulties when switching from block-based to text-based programming [28]. Students perceive block-based programming as not "real programming" when they switch to text-based programming. Furthermore, block-based programming cause students to show bad programming habits, making it difficult to switch to text-based programming [29]. The

inconclusive results of the above studies suggest the need for further studies on the effect of block-based programming interfaces on students' perceptions of programming.

1.3. Purpose

This paper proposes a symmetry language of block-based called JLCoding, to solve the difficulties encountered by novice programmers in the past from block-based to text-based programming and to make it easier for non-programmers to learn text-based programming. The language combines the advantages of text-based and block-based programming. In [30], the definition of symmetry is "In computer science, one type of symmetry is if the features of a programming language operate equally on all of the things in the programming language". Hence, JLCoding is a transition from block-based to text-based programming. Its interface and functions are similar to Scratch. The system contains graphic teaching to teach the basic knowledge of programming so that students can maintain interest and confidence when learning computational thinking. Additionally, JLCoding uses the principle of English code, which is similar to text-based programming to allow the students to be familiar with the correct programming environment. The JLCoding system provides many templates and examples to demonstrate the basic knowledge about logic, control, and flow, which are illustrated with a block-based graphic and text-based structure.

Thus, the purposes of this study are shown below:

- Design a programming language to help the student transits from block-based to text-based programming.
- Help the students who have never studied any programming to easily learn text-based programming.
- The proposed system JLCoding can achieve the purposes with the advantages:
- Easy to learn: The students who have no programming experience can easily learn the basic knowledge from JLCoding.
- Easy to transmit from the block-based to text-based programming skills. The students who learn the block-based language can convert their experience to a text-based environment quickly.
- Quick to produce diversity projects. JLCoding allows the student to generate a different kind of produce.
- Provide object-oriented programming. Students can combine complex logic rules.
- Avoid debugging problems in block-based programs. JLCoding provides a friendly debugging solution that allows the students to find the errors.
- Easy to link up with other high-level programming languages such as Python, Java, and other text-based programming in the future.

To test the performance of the JLCoding, this study compares the experiences of beginners when they learn block-based programming and JLCoding. Further, we investigate the learning effectiveness, interest, and confidence of novice programmers learning text-based programming through JLCoding. Thus, this paper is divided into sections. In Section 2, we describe a brief previous work in the field. Section 3 describes the research methodology. The study surveys to answer this hypothesis. Section 4 shows our experiments, results, and discussions. Section 5 describes the conclusions and future work of the study.

1.4. Terminology

The following terms are frequently used throughout this paper and their meanings are discussed when relevant. For clarity, a summary of definitions is provided here:

- Text-based programming: A general term for program editing software in the general sense, which is used for program design by inputting text. It is then debugged and compiled to produce programs that make the computer work. Although it is possible to generate programs with various functions, it is also difficult to learn. Some famous text-based programs are Java, C++, and Python.
- Block-based programming: A program editing software that does not require text input. The user only needs to drag the blocks to make combinations to generate the

program. Although the works it can produce are limited to a certain range, it is favored by novice programmers because of its ease of learning and operation. An example of a block-based program is Scratch.

- JLCoding: A programming environment that combines the ease of learning of block-based programming with the unrestricted functionality of text-based programming. While facilitating the creation of novice programmers, it also maintains the diversity of the works produced.
- N: the total number of test samples.
- Mean: the average of the score.
- SD: the standard deviation of the score.
- SE: the standard error of the score.
- SEM: the structural equation modeling.
- t : the test quantity.
- df: the degrees of freedom.
- p : the significance of the hypothesis.

2. Related Work

2.1. Introductory Programming Difficulties

Introductory programming is considered an important foundation in the field of computer science [31]. However, many novice programmers have difficulty with this subject. Difficulties are not only experienced by students at the secondary school level, but also by first-year college students. The number of students who experience attrition in the first year is very high [32–34]. Some researchers show that nearly a third of students fail or drop out of this course [20,35,36]. The reasons for failure or dropout vary, ranging from motivational factors, teaching methods, heavy demands, and inability to catch up [37–39].

2.2. Block-Based Programming

In recent years, block-based programming has taken an established position in computer science education [29,40,41]. Block-based programming is becoming increasingly popular not only in education [34] but also among end-users [40]. By removing many difficulties common to novice programmers [20], this programming environment can attract novice programmers of all ages, ranging from elementary school students to college students [41,42]. This programming environment is usually provided as an introduction before students are faced with more difficult text-based programming [43].

Block-based programming provides puzzle pieces for clues as to how and where commands can be used. Programming activities are then similar to assembling a puzzle, where the user drags the desired blocks into the coding area and combines them to form a script. When two blocks cannot be combined to form a valid syntactic statement, the programming environment prevents it from making mistakes but maintains the practice of assembling instructions sequentially [10,41]. The use of block-based programming has many benefits [34,42], such as reduced cognitive load [44], increased understanding of programming structures [45], and increased efficiency in completing programming tasks [27].

Although it has been firmly accepted in educational settings, there are still challenges regarding students' perceptions of the role of block-based programming in introductory programming. The main question is whether block-based programming means "real programming" [41]?

The biggest challenge that most novice programmers face is an understanding of the fundamentals of programming, which will enable them to develop algorithmic and computational thinking in order to be able to design programs [22], and simultaneously prevent misunderstandings and different prejudices against science education [25]. Students must have confidence that they are using computers not only as users but also as creators [46].

Problem-solving skills are an integral part of understanding proper programming concepts [47]; however, most students who study introductory programming tend to

develop superficial knowledge and fail to create problem-solving strategies by using programming constructs [48]. Students without a programming strategy, such as using loops appropriately in a program, will have difficulty combining conditionals and loops to provide a viable solution [49].

Previous studies in science education have shown the types of problems students experience when they face text-based programming, among others, control structures such as variables and other data structures, various types of looping structures, conditional use logical flow, and Boolean logic [50–52]. Departing from the difficulties of novice programmers, [9] conducted research on how block-based programming, such as Scratch, relates to text-based programming, such as Java, C++, and Python. Half of the students were selected to complete student worksheets that showed a relationship between Scratch, Java, C++, and Python. Their findings showed no significant difference between students who completed and those who did not complete the worksheet in terms of beliefs and perceptions; however, a study [8] that investigated misconceptions about loops, variables, and Boolean logic found different results. Their results showed that students are generally unfamiliar with the use of variables and harbor misconceptions about these variables. Nevertheless, they also run into problems with how loops work and Boolean operators.

2.3. Confidence in Programming Abilities

Block-based programming is a programming environment that is taught as introductory programming for novice programmers. A programming environment such as this has succeeded in lowering programming barriers and making it easier for students to start programming [29,53,54]. Nevertheless, the mastery of text-based programming remains the standard for advanced and professional programmers. This gap between the two programming styles creates difficulties in teaching programming at the school level [53,55]. Additionally, this gap also has an impact on students' low self-confidence in their programming skills [29,45].

Several researchers have conducted research related to aspects of self-confidence in programming to [56] determine whether the basic concepts of programming could be effectively taught to students with limited or no programming backgrounds. The results of their research on students who used Alice showed an increase in attitude scores, but the increase was not statistically significant. [57] Extending their research by investigating the self-confidence aspect of attitudes, their results showed a significant increase in self-confidence in their programming; however, their hypothesis that students who work in pairs will be more confident than those who work individually is not significant. [13] We compared the attitudes and learning outcomes of sixth-grade students in programming using both Logo and Scratch. They hypothesized that students studying Scratch would have a more positive attitude toward programming. Contrarily, their findings showed that students who studied Logo on average had higher confidence in their ability to program [10,41]. Compared block-based and text-based programming in introductory programming classes in secondary schools. Their findings showed there was no significant difference between students in the two conditions concerning self-confidence or pleasure. Furthermore, students in block-based programming saw a significant increase in their confidence in their programming skills. Such a thing is not found in students in text-based programming.

2.4. Scratch

In recent years, there has been a shift in the computational curriculum from focusing on information and communications technology to broader ones such as informatics, digital literacy, and computer science [58,59]. The aim of this shift is for students to master the skills and knowledge needed in an increasingly digital world, with learning content that focuses on computer programming, robotics, and computational thinking [60].

The most widely used educational programming tool is Scratch. Scratch has been used by more than 53 million projects since its launch in 2007 [61] and has become the most popular programming environment in basic education [62].

In Taiwan, Scratch is the most popular block-based programming because it supports the Chinese language and provides a lot of teaching materials; therefore, the education bureaus of many cities in Taiwan have assigned Scratch as the recommended tool to learn the basic knowledge of science and technology. For example, Taipei city has expressed Scratch as the proper tool to teach the elementary or junior high school students in the syllabus of information technology courses for elementary schools in the science and technology field in Taipei City. There are more than one million users in Taiwan that use Scratch to develop projects; therefore, this study selects Scratch as the template to compare with the proposed language.

Scratch uses block-based programming, similar to a jigsaw snippet. This programming implements a block-like structure in which each block is different in shape and color to provide clues as to how instructions can be structured and to differentiate between concepts [63]. In addition, this programming language also encourages students to create media-rich content according to their desires [29].

The coding area in the center of Scratch's screen is the main area that connects the block palette on the left-hand side and the stage area on the right-hand side. The user only needs to put the character into the stage area, then drag the code block from the block palette into the coding area. Finally, click the green flag (compile) to execute the program to control the characters in the stage area. The interface of Scratch is shown in Figure 1.

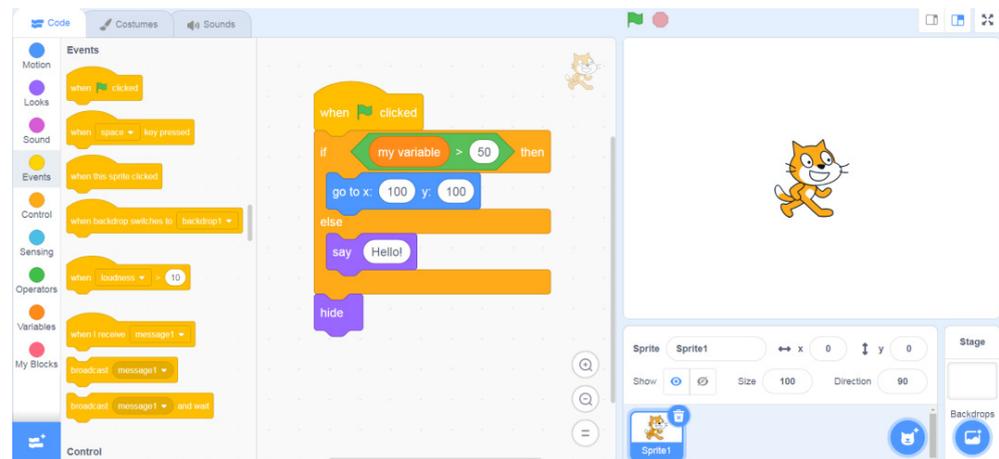


Figure 1. The interface of Scratch.

The blocks in the block palette have different colors for convenient memory and discrimination. It contains Motion, Looks, Sound, Events, Control, Sensing, Operators, Variables (Lists), and Function Blocks—as shown in Figure 2. Users can generate different types of programs by putting different types of blocks into the coding area for combination. Each block can be treated as a puzzle. The program can be executed when the user can successfully combine it with other blocks. The definitions of each block are shown below:

- Motion: Control the position, angle, rotation, and movement of the character.
- Looks: Control the shape, color, size, and special effects of the character, and display the text.
- Sound: Control the sound playback and volume.
- Events: Set the program to be executed when some event is triggered.
- Control: Set conditions and loops.
- Sensing: Obtain mouse and keyboard information, distinguish touching.
- Operators: Set logic operator, arithmetic operator, string operator, and obtain a random number.

- Variables: Generate variables to store the information.
- Function Blocks: Customized block.

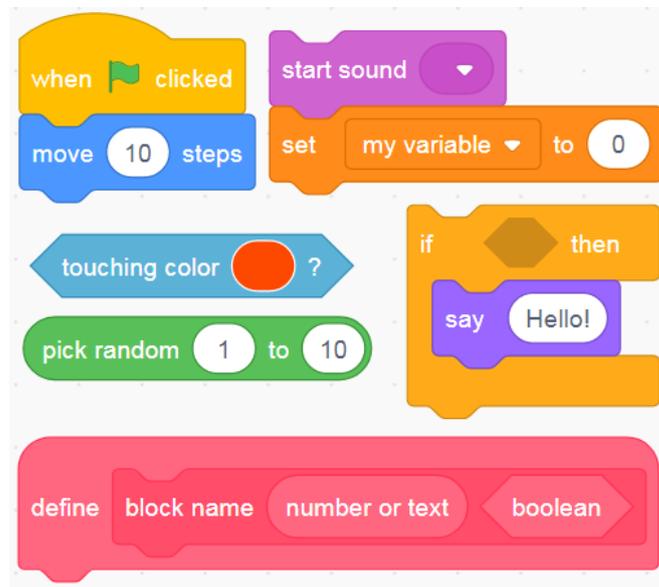


Figure 2. Different types of Scratch blocks.

The stage area is used to set characters and display the results of the program. The user can set the name, coordinates, size, and angle of the character in the area. Scratch is object-oriented programming. Every character has its coding area to prevent all programs from being set in the same area. Figure 3 shows an example program of the apple. The user can switch the size of the stage area, and the green flag button is “Compile” to execute the program, while the red button is used to stop the program.

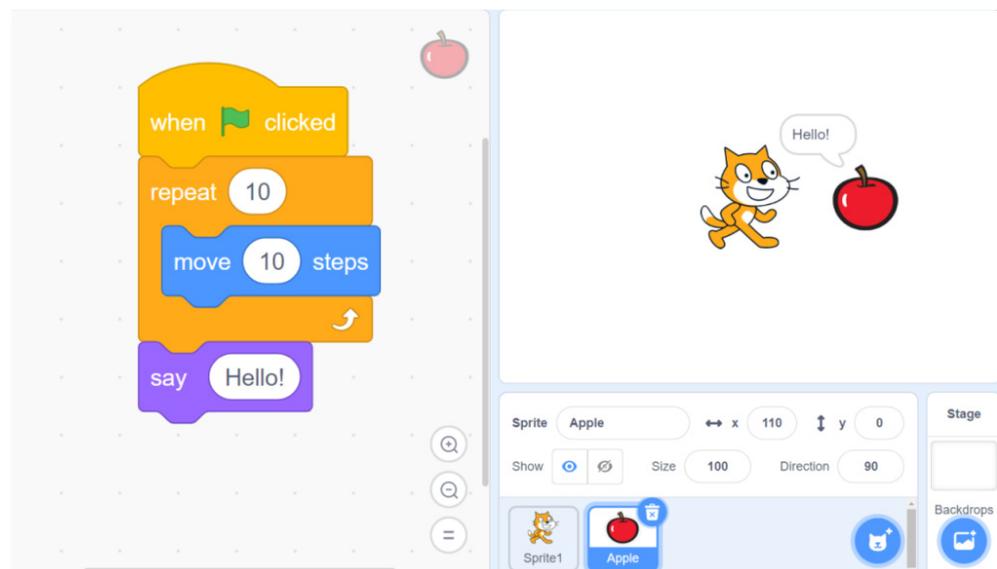


Figure 3. The stage area and object-oriented nature of Scratch.

3. JLCoding

Compared to Scratch, which has a longer history, JLCoding is a programming language that is still new. JLCoding was launched in 2017 and is an open-source language that combines the interactivity and syntax of “scripting” languages, such as Python, Matlab, and R, but with the speed of “compiled” languages such as Fortran and C [64]. Much

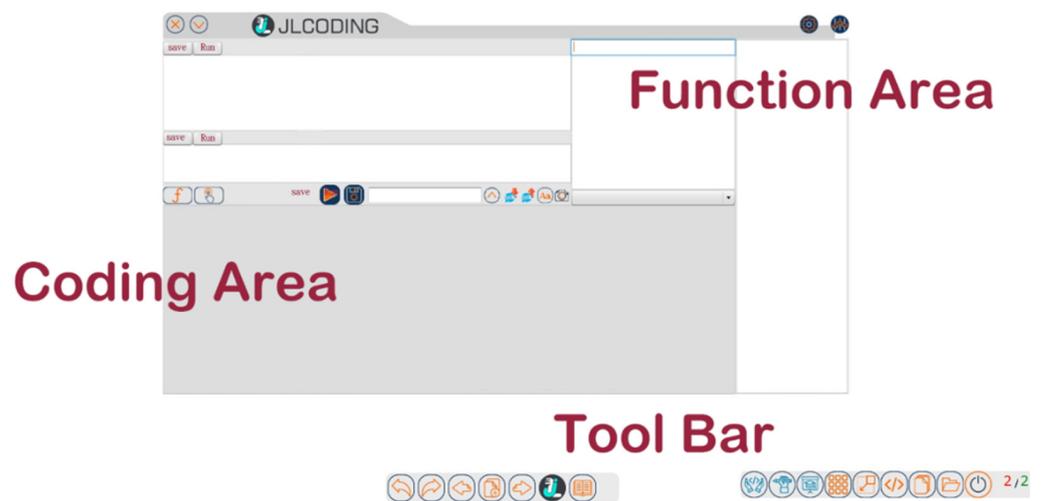
of the motivation for the development and use of JLCoding is devoted to solving two language problems that both beginners and professional programmers encounter. In scripting languages such as Python, the user types the code line by line into the editor, and the language interprets and runs it, then returns the result immediately; however, in languages such as C and Fortran, code must be compiled before it can be executed. Scripting languages such as Python are easier to use, whereas languages such as C and Fortran produce faster code. As a result, programmers often develop algorithms in a scripting language, then translate them into C or Fortran. JLCoding can solve the two language problems because it runs similar to C but reads similar to Python [64–66].

3.1. Environment of JLCoding

Figure 4 shows the starting view of the JLCoding. The beginning of JLCoding is the object. The system includes many examples, functions, and templates to help the students quickly understand the logic of coding. The coding environment of JLCoding is shown in Figure 5.



Figure 4. The starting view of JLCoding.



(a) Coding window of JLCoding

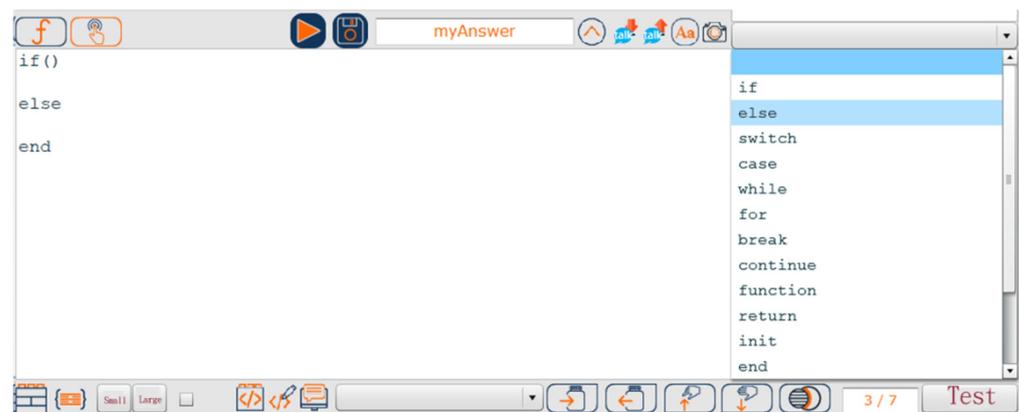
Figure 5. Cont.



(b) the stage area of JLCoding

Figure 5. Coding environment of JLCoding.

The coding area in Figure 5a is the main area to type the codes. The function area in Figure 5a gives the entire function used in the coding area. This method finds the keyword from the function area. The code structure of the keyword will be shown in the coding area. The stage area in Figure 5b shows the running results of the codes. For example, in Figure 6, we select the keyword “if” from the function area. The coding area will automatically show the structure of “if”.

**Figure 6.** Keywords in the function area.

3.2. Coding Example of JLCoding

According to the definition of a 12-year national basic education syllabus in Taiwan for junior high school students to learn computer science courses, the students need to understand computer data forms, variables, input/output, arithmetic operations, logical operations, selection structures, and repetition in the field of programming; therefore, this study considers all requirements in the syllabus to design the JLCoding language. After learning the courses, the student has the basic knowledge required to write the program. Typically, the structure of JLCoding is similar to a text-based program such that the student can learn the basic logical structure of text-based programming at the same time.

This sub-section shows some coding examples by using JLCoding and compares them with the coding environment of Scratch.

3.2.1. Set Variable

The first example demonstrates how to move the object—a bee—from left to right. In Scratch, the block functions “set x to” and “set y to” are used to move the bee from the position (0, 0) to the position (100, 100)—as shown in Figure 7.

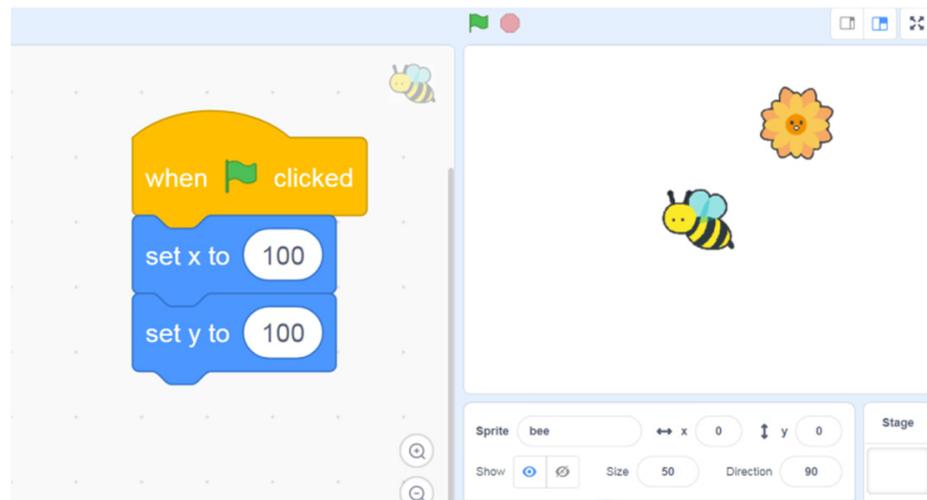


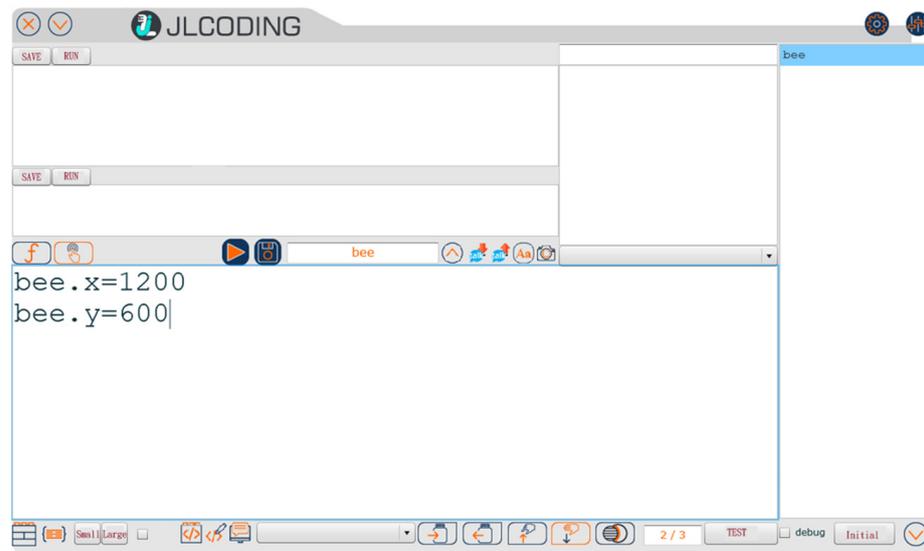
Figure 7. Move a bee from left to right by using Scratch.

Instead of the block function, JLCoding uses the object-orient concept to design the attribute of the object. Figure 8a shows the beginning position of the bee. The initial position of the bee is set to be (900, 200). JLCoding uses “bee. x” and “bee. y” to set the position of the object bee. Figure 8b shows how to code the moving operator in the coding area. Figure 8c is the final running result.



(a) Beginning position of the bee

Figure 8. Cont.



(b) move the bee to the position (1200, 600)



(c) the final position of the bee

Figure 8. The coding example of JLCoding to move the bee.

3.2.2. Input and Output

In Scratch, the event block creates a way to interact with the program and the user. The user uses the mouse and keyboard to control the object. The square in the code area on the left of Figure 9 indicates that when the space key of the keyboard is clicked, the program says “Hello!” for 5 s. The user uses the space key as the input to control the program execution, and the text will appear on the upper right of the duck for 5 s to achieve the output.

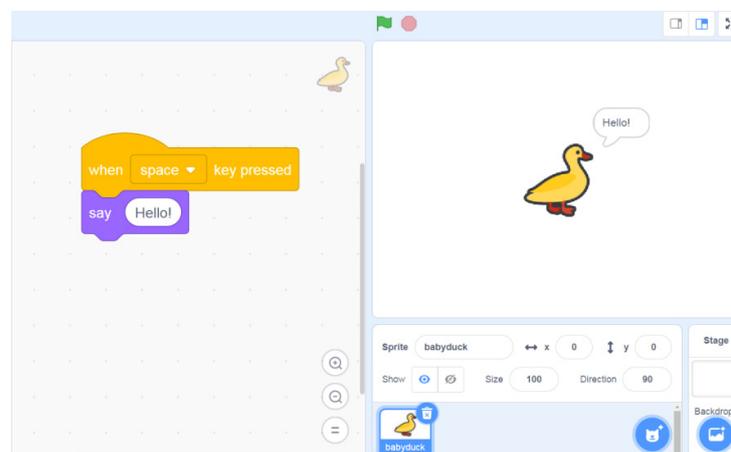


Figure 9. Input and output by using Scratch.

On the other hand, JLCoding does not use the events box for input; it uses buttons instead. Users can interact by setting the object as a button and entering the program that will be executed when the button is pressed in the coding area. The example in Figure 10 shows that after clicking the duck button, the program says “Hello!” for a period of time. When the user presses the button with the mouse, the text will appear above the duck as output.

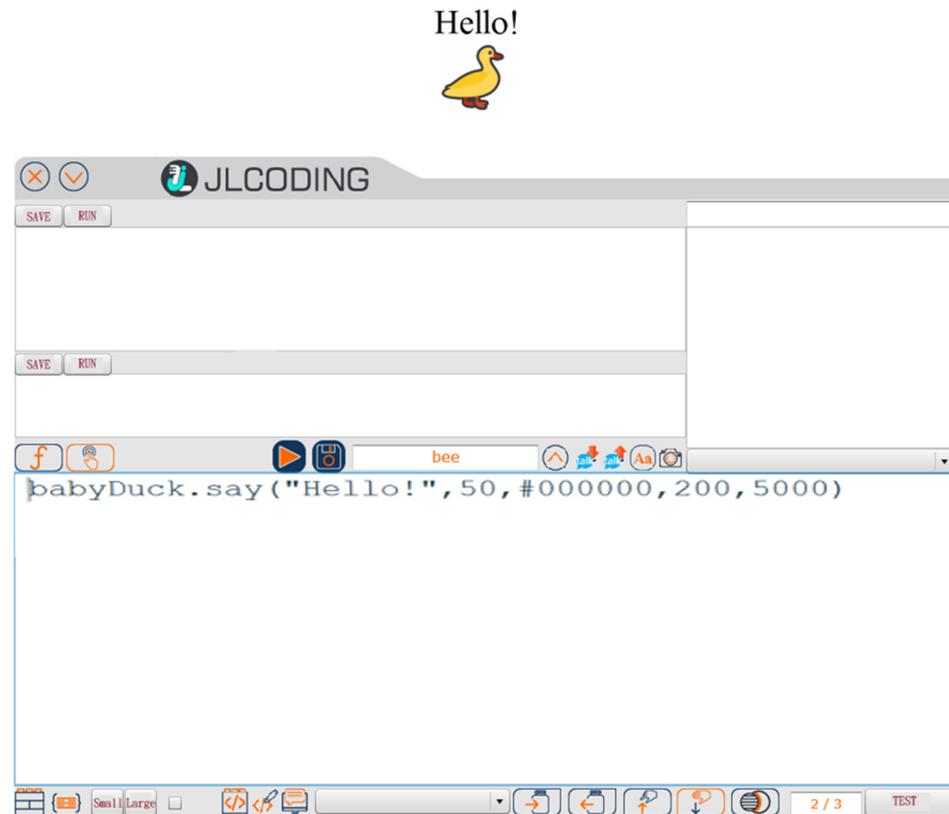


Figure 10. Input and output by using JLCoding.

3.2.3. Arithmetic

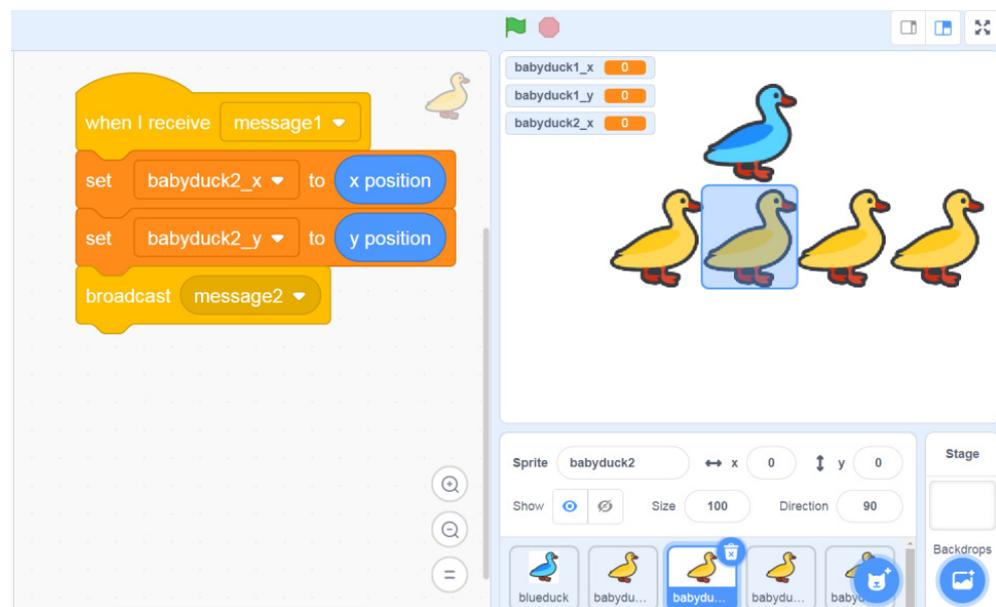
Figure 11 shows an example of how to calculate the required variables in Scratch. The program moves the blue duckling to the back of the duckling team while maintaining the same distance as the other ducklings. The code in Figure 11a shows the code in the fourth duckling that obtains the X-coordinate and Y-coordinate of the duckling. After obtaining the coordinates of the fourth duckling, use “message1” to drive the code in Figure 11b to obtain the coordinates of the third duckling. Similarly, “message 2” is performed after the coordinates are obtained to drive the code in Figure 11c to calculate the moving position of the blue duckling. Figure 11c will calculate the X coordinate as (the 4th duckling’s X coordinate)—(the distance between the 3rd duckling’s X coordinate and the 4th duckling’s X coordinate), and the Y coordinate using the same coordinates as the 4th duckling. When the calculation is complete, the “go to” block function will move the blue duckling to the calculated coordinates, as shown in Figure 11d.

Compared with Scratch, the code is much easier to use in JLCoding. Since the coordinates of all objects have already been acquired, the user only needs to input the calculation formula into a coding area to calculate the position of the blue duckling. The code is shown in Figure 12. The algorithm in JLCoding is almost the same as Scratch. The program sets the X coordinate of the blue duckling to be (the X coordinate of the 4th duckling)—(the difference between the X coordinate of the 3rd duckling and the X coordinate of the 4th duckling), and the Y coordinate of the blue duckling will be the same as that of the 4th

duckling. The execution results are shown in Figure 13, in which the blue duckling moves to the correct position.

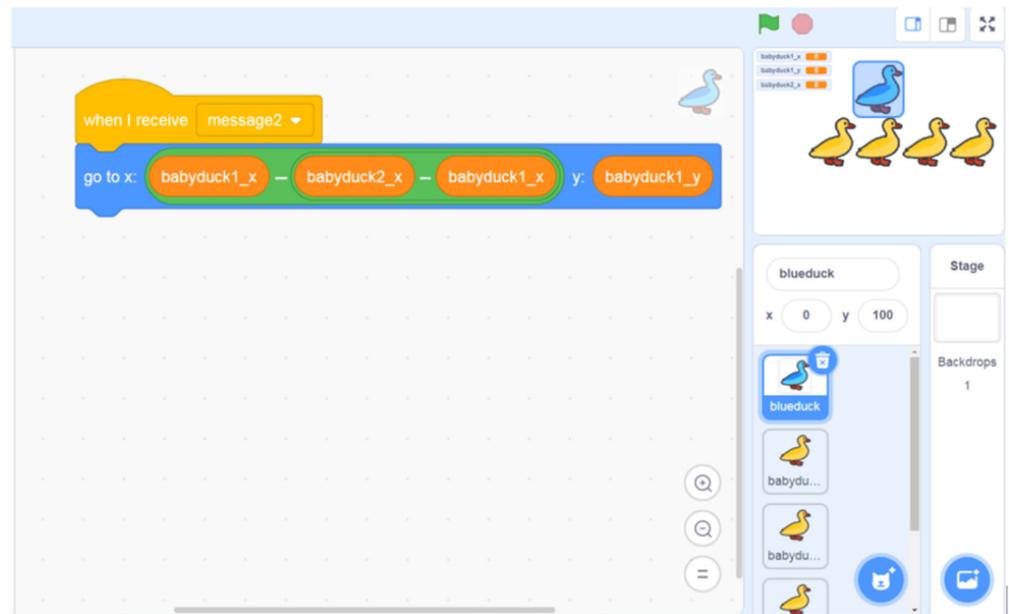


(a) Obtain the position of the fourth duckling

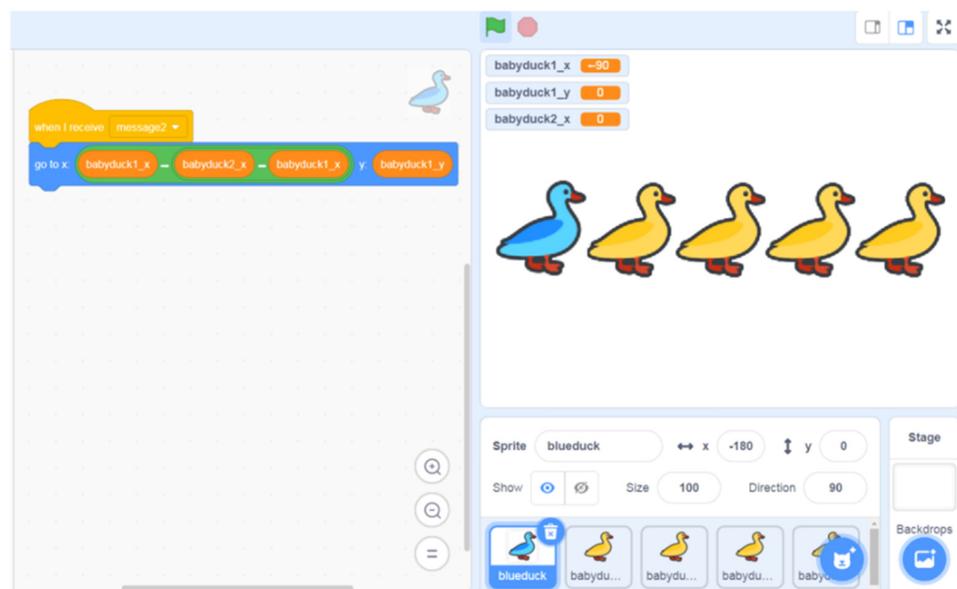


(b) obtain the position of the third duckling

Figure 11. Cont.



(c) calculate the position of the blue duckling



(d) the results

Figure 11. Arithmetic by using Scratch.

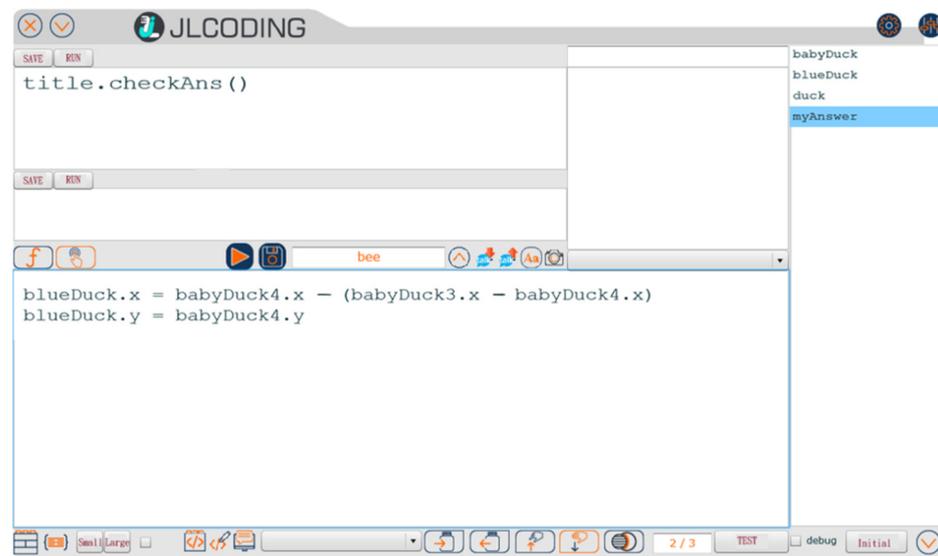
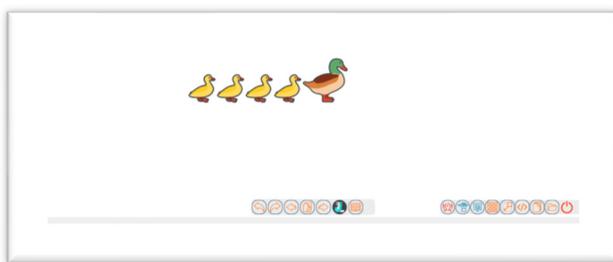
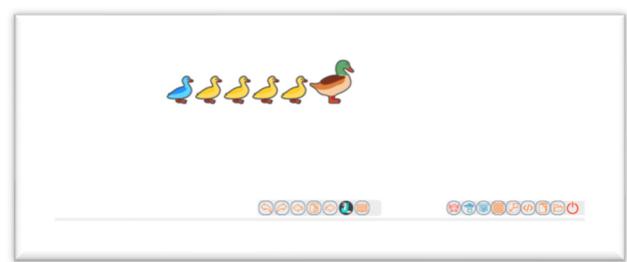


Figure 12. Arithmetic in JLCoding’s coding area.



(a) blue duckling in the wrong position



(b) blue duckling set in the correct position

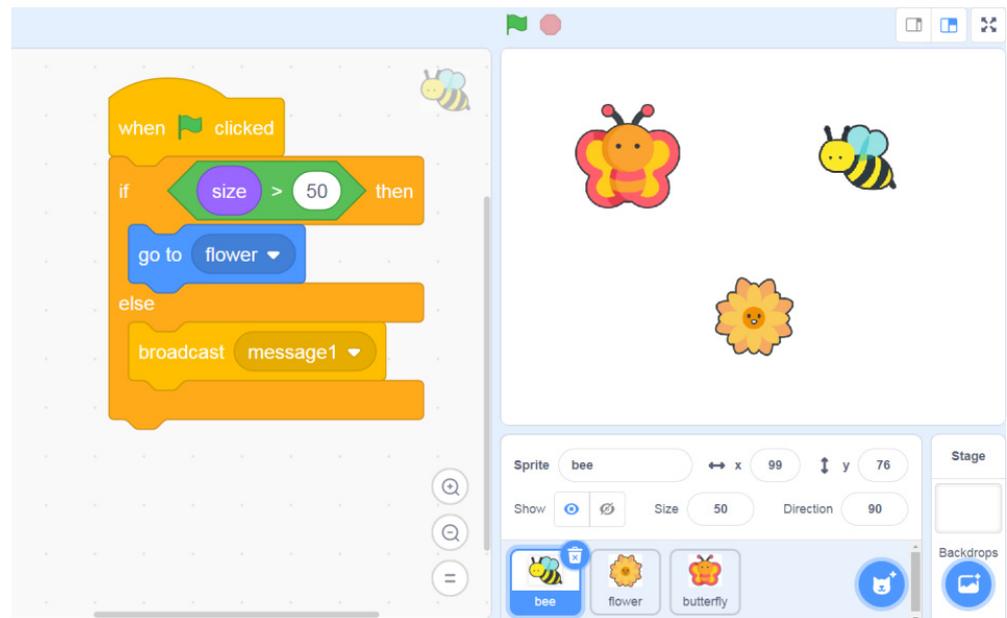
Figure 13. Arithmetic by using JLCoding.

3.2.4. Conditional

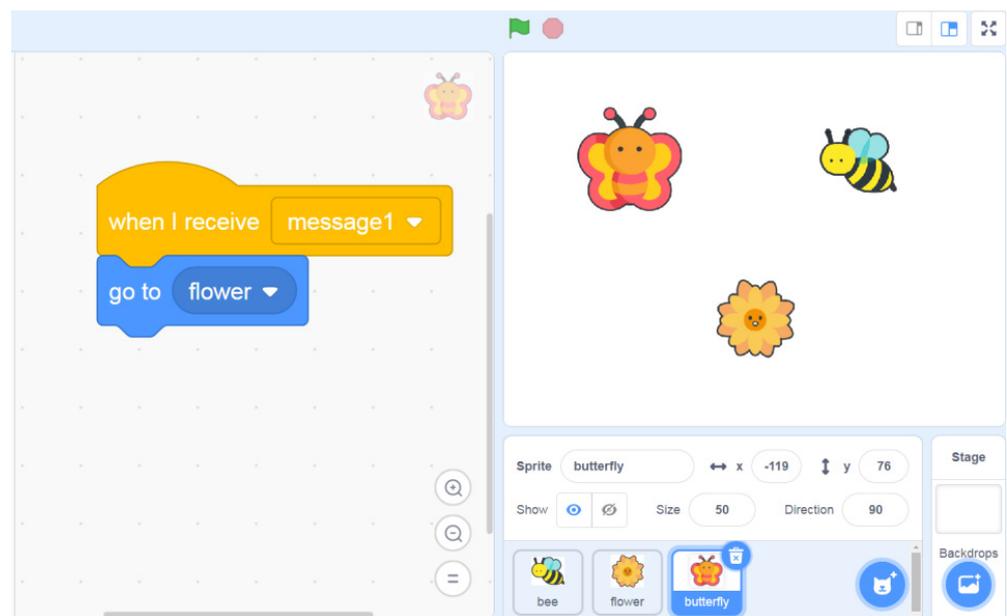
Conditional is a selection structure that determines the items to be executed according to the conditions specified in the code. Figure 14 shows a conditional decision to decide whether the bees or the butterflies will be moved to the flower.

The code in Figure 14a uses an “if/else” block as the main body and makes choices based on whether the size of the bee is greater than 50. If the condition rule is true, then “go to flower” will be executed. The bee will move to the flower. Otherwise, it will choose not to execute the go to block of the bee and only execute “broadcast message1” to drive the code of the butterfly in Figure 14b to move the butterfly to the flower. The final result is shown in Figure 14c.

In the JLCoding, the user inputs three strings, “if”, “else” and “end”, as the structure. The selection condition rule ($\text{bee.width} > 50$) is used to decide whether the bee or the butterfly will move to the flower. According to the “if it is true” of the selection condition, choose to use “bee.moveTo(flower)” to move the bee to the flower, or choose to use “butterfly.moveTo(flower)” to move the butterfly to the flower. Figure 15 shows the codes and Figure 16 shows the execution results.

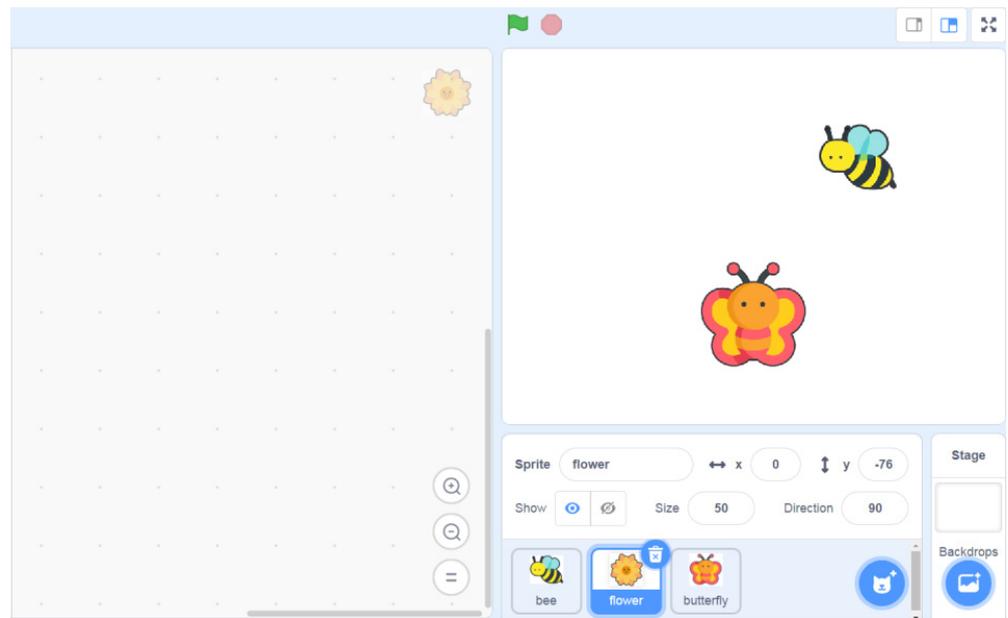


(a) the code of the bee



(b) the code of the butterfly

Figure 14. Cont.



(c) the results

Figure 14. "If" condition rule to move the bee or the butterfly to flower by using Scratch.

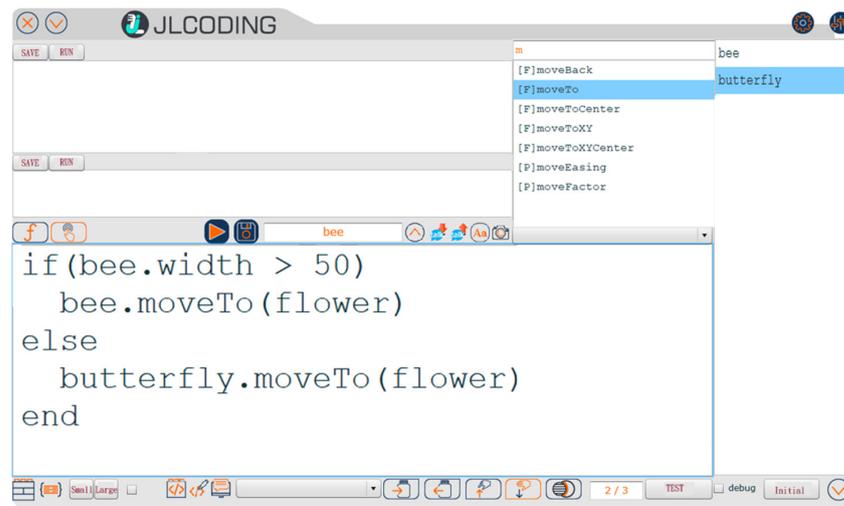
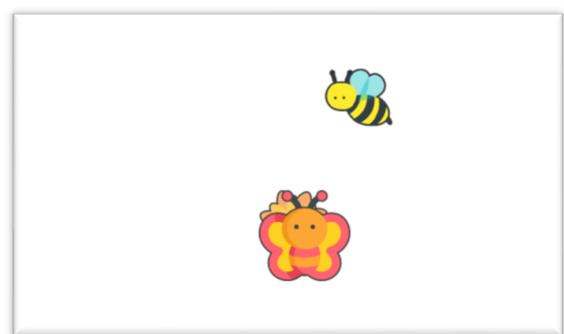


Figure 15. Conditional in JLCoding's coding area.



(a) before



(b) after

Figure 16. Select to move butterfly to flower by using JLCoding.

3.2.5. For Loop

The second example demonstrates how to generate several objects by using the “For loop.” In the Scratch environment, the block function “repeat” is used to perform the “For loop” operator. For each loop, the system generates one duck and moves it to a different position. Figure 17 shows the “repeat” example using Scratch.

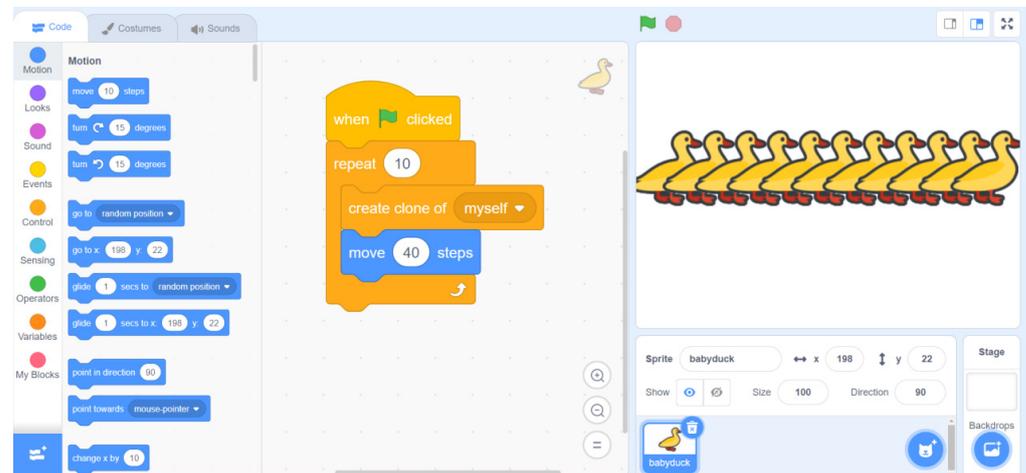


Figure 17. Generate 10 ducks by using Scratch.

Similar to Scratch, JLCoding uses the For loop structure, which contains for and ends keywords to perform the “For loop” operator, which is the same as the text-based language. The user types the keyword “for” in the function area, and JLCoding will generate the structure automatically. JLCoding uses “clone” to generate a new object and uses “object.x” to change the position. Figure 18 shows the running results using JLCoding.

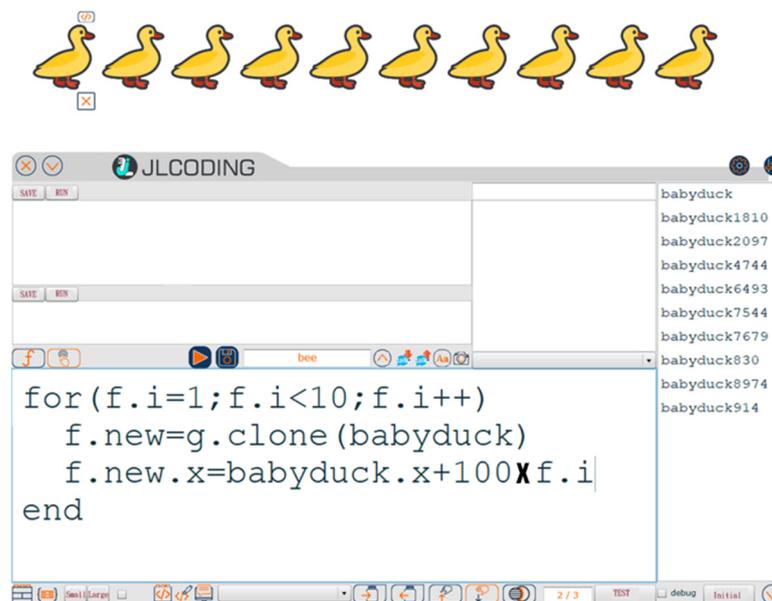


Figure 18. Generate 10 ducks by using JLCoding.

3.3. Features of JLCoding

JLCoding is a programming language that helps the student transform from blocks to text-based programming. To avoid rejection by students who are learning text-based programming for the first time, JLCoding uses the concept of object-oriented programming similar to Scratch. Students put the character on the stage area and use the program to control it. Similar to Scratch, the stage area of JLCoding is the whole screen. An additional

window is added to the coding area to prevent the coding area from squeezing the stage area. Thus, students can make more creations in the expanded stage area. The diagrams are shown in Figures 19 and 20.

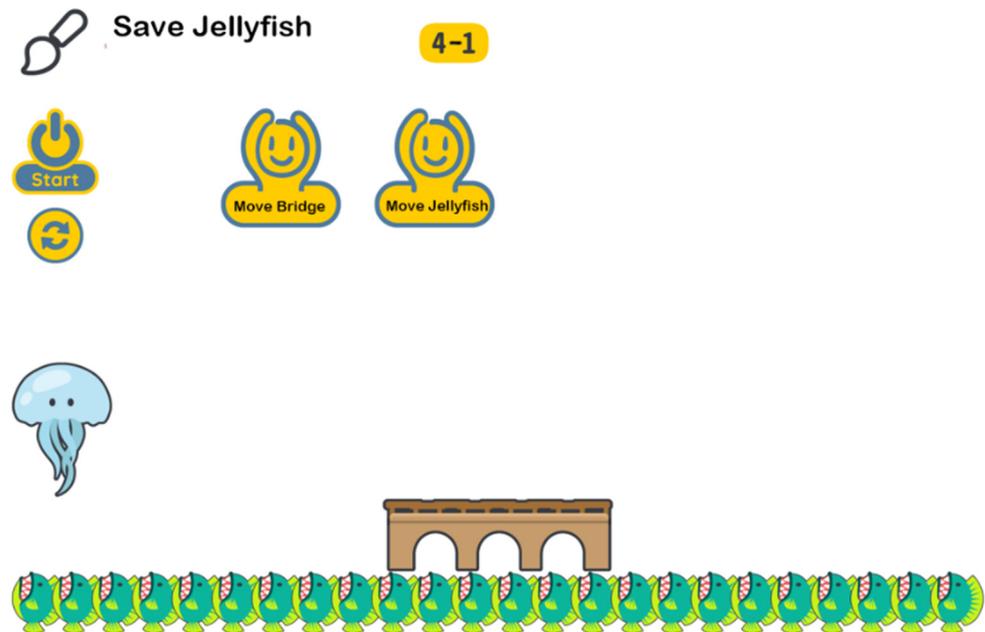


Figure 19. The stage area of JLCoding. The user needs to move the bridge and jellyfish to save jellyfish.



Figure 20. The coding window in front of the stage area.

Instead of dragging the block to generate the program, JLCoding allows the students to type some codes in the coding area to make students acquainted with the programming environment of text-based programming. To reduce students' difficulty in memorizing codes, the function area provides hints to help students enter codes. Then, the students click the triangle button in the middle of the coding area to compile and execute the program.

Furthermore, JLCoding has built-in teaching courses and practice questions to prevent students using the system for the first time from having no idea how to start the programming procedure. The students can learn by themselves without using additional software. The course content includes object naming, coordinate settings, length and width settings, hexadecimal color code, object movement, rotation, collision, container, copy, and other functions, as well as learning how to use functions, variables, expressions, and conditional expressions. The course outline is shown in Table 1. Each course uses vivid pictures to illustrate the learning objectives and allows students to think of how to use the functions they learned when they practice after class. The textbook screen is shown in Figures 21 and 22.

Based on the above description of JLCoding and Scratch, the comparison between the two programming languages is shown in Table 2.

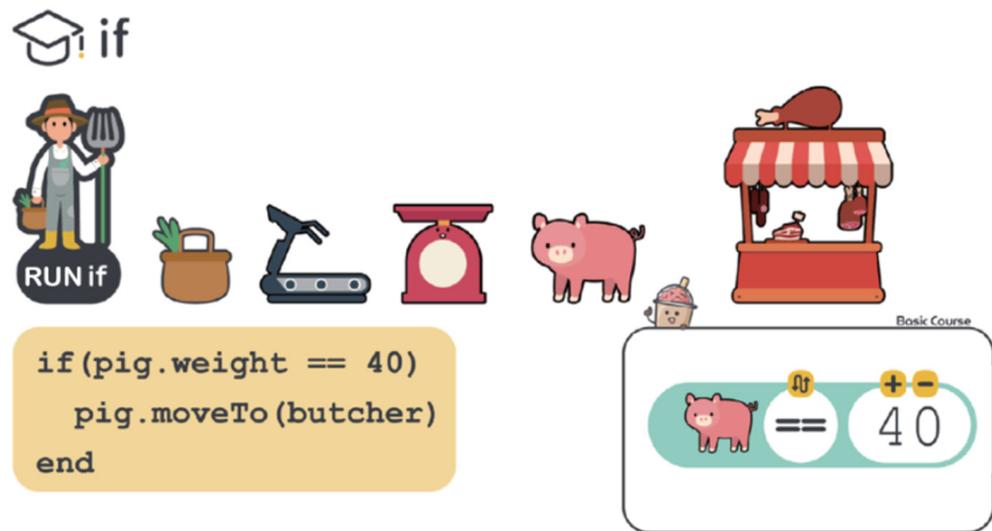


Figure 21. Built-in teaching of JLCoding.

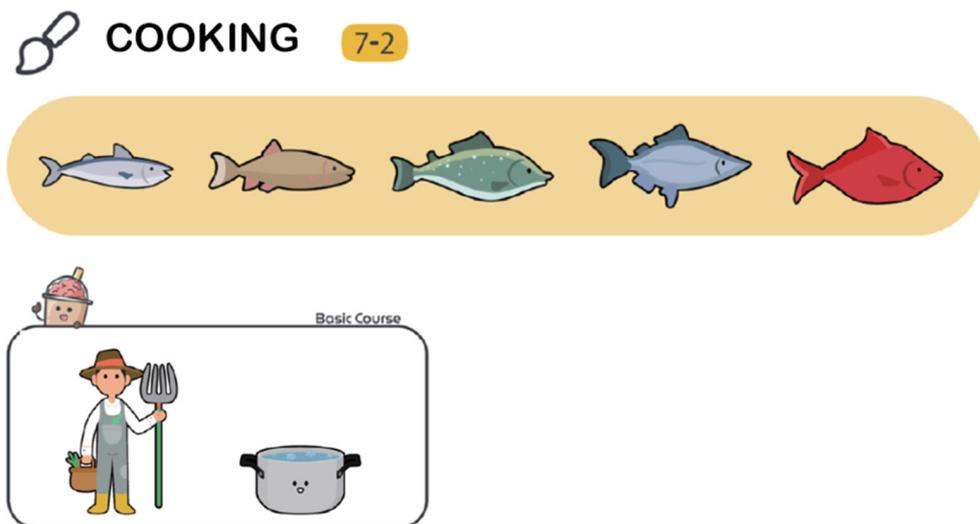


Figure 22. Built-in practice questions of JLCoding.

Table 1. Course outline of JLCoding.

Course	Theme	Content
Basic Course 1	Little blue duck leaving the team	1. Understand JLCoding and system. 2. Learn how to set the position and coordinates of objects.
Basic Course 2	Pigs will set sail	1. Learn how to set the length and width of object. 2. Learn how to set the sizes of objects.
Basic Course 3	Missing turtle eggs	1. Learn how to set the color of objects. 2. Forward and turn of objects.
Basic Course 4	Trapped jellyfish	1. Movement and rotation of objects.
Basic Course 5	Infinite bomb	1. Learn how to use the code for copying objects.
Basic Course 6	Pirate's gold coin	1. Learn how to set variables. 2. Arithmetic operations of variables.
Basic Course 7	Delicious fish soup	1. Learn how to use the code for If/else.

Table 2. The comparison between Scratch and JLCoding.

Item	Scratch	JLCoding
Entry barrier	The difficulty of getting started is extremely low. Students can execute the program and see the effect without typing the code and dragging the building blocks directly with the mouse; however, when there are relatively problems, it is difficult to debug, and it is inconvenient to perform complex combinations of logical conditions.	There is a certain degree of difficulty in starting, but JLCoding redefines the original complex code into modern code that is easy to understand. With the program assistance function that comes with the JLCoding platform, it can greatly reduce the cost of learning to write code.
Creative space	The number of functions is limited. The limitations of the platform compress the space for creation. The student cannot freely create their own function.	A platform specially designed for teaching allows students to freely use a variety of materials and hundreds of rich built-in functions (APIs) to support students in designing games, APPs, small systems, etc. The platform can be connected to hardware or AI. Students' creative field further expands to the Internet of Things (IOT) or artificial intelligence.
Teaching support	The design focuses on easy operation for students and supports a few necessary teaching auxiliary functions for teachers. When teachers use the program platform to teach, they usually need to open other auxiliary tools such as PPT.	JLCoding integrates teaching, learning, and practice. Teachers use the courseware and built-in auxiliary tools to teach on the platform to improve the teaching effect. Students use the student courseware to learn and can also practice directly on the platform, improving learning efficiency and effectiveness.
Programming environment	What you see is what you get (WYSIWYG), so that students can quickly see the results of the program; however, there is still the problem of separation of teaching, learning, and practice, which reduces the learning effect.	JLCoding achieves WYSIWYG and the integration of teaching, learning and practice. The teaching materials and courseware can be switched between teaching and learning smoothly without switching any software, and the programming environment has function prompts and detailed Chinese explanations. Remove language and format barriers.

3.4. Compare with Khan Academy

Khan Academy [67] also provides some training courses for computer programming learning. At the Khan Academy, JavaScript and HTML languages are the most popular programming courses. The platform is a text-based programming training environment. There are many lessons and exercises on the platform, and they also provide videos. For formal programming, it may be a good idea to use the videos to learn some of the text-based programming training; however, it requires the user to pay attention to more details, such as typing relative parentheses and noticing the placement of semicolons [68]. The

users might easily forget the main learning content, which in turn can contribute to one losing their self-confidence to learn. The design of JLCoding allows users to focus on the logical thinking of programming without paying attention. The respective advantages are compared in Table 3.

Table 3. Compare Scratch, JLCoding, and JavaScript.

Item	Scratch	JLCoding	Khan Academy (JavaScript)
1. Programming environment	Block-Based programming	Text-Based programming	Text-Based programming
2. Programming type	Drag the blocks	Typing string	Typing string
3. Creation	Programmatically control the placed objects	Programmatically control the placed objects	Programmatically generate controllable objects
4. Programming paradigm	Object-Oriented Programming	Object-Oriented Programming	Functional Programming
5. Memorizing codes	Find in block palette	Find in functional area	Find in coding documentation
6. Separators	Not using curly braces and semicolons	Not using curly braces and semicolons	Need to be used correctly
7. Teaching	Not provided	Built-in Graphic teaching	Built-in video teaching
8. Practice	Free creation	Built-in practice questions	Built-in practice questions

4. Research Methods

4.1. Research Object

To understand students' learning status and satisfaction with JLCoding, this study selected secondary school 7th grade (12–13 years old) students for testing; however, due to the limitations of school teaching equipment and overall teaching curriculum planning, only 41 students were selected for the experiment. The students in this class understand both Chinese and English, and all of them have basic English typing skills. All of them agreed to the experiment and understood the entire experimental process.

4.2. Research Process

The flowchart of this research is shown in Figure 23. The experiment first went through a "pre-test" process to understand the basic abilities of the students; then, we taught JLCoding for seven weeks. Afterward, we conducted the "post-test" to analyze the results of the pre-test and the post-test.

Consequently, the background information of the students was collected during the pre-test in this study, and the JLCoding test was performed simultaneously. The JLCoding test was used to obtain the students' understanding of programming knowledge before taking the JLCoding course and was recorded in the form of scores. Then, the students learned JLCoding for 2.5 h per week. The courses are listed in order.

After completing the training courses, the students performed a post-test to evaluate the effectiveness of the learning. The post-test questionnaire contains the same JLCoding test as the previous test, and a text-based programming quiz is added to determine whether JLCoding can connect to text-based programming. Finally, this study uses the "Programming Confidence Questionnaire" to understand the degree of recognition of the students with programming abilities and asked whether they were willing to continue improving their programming abilities.

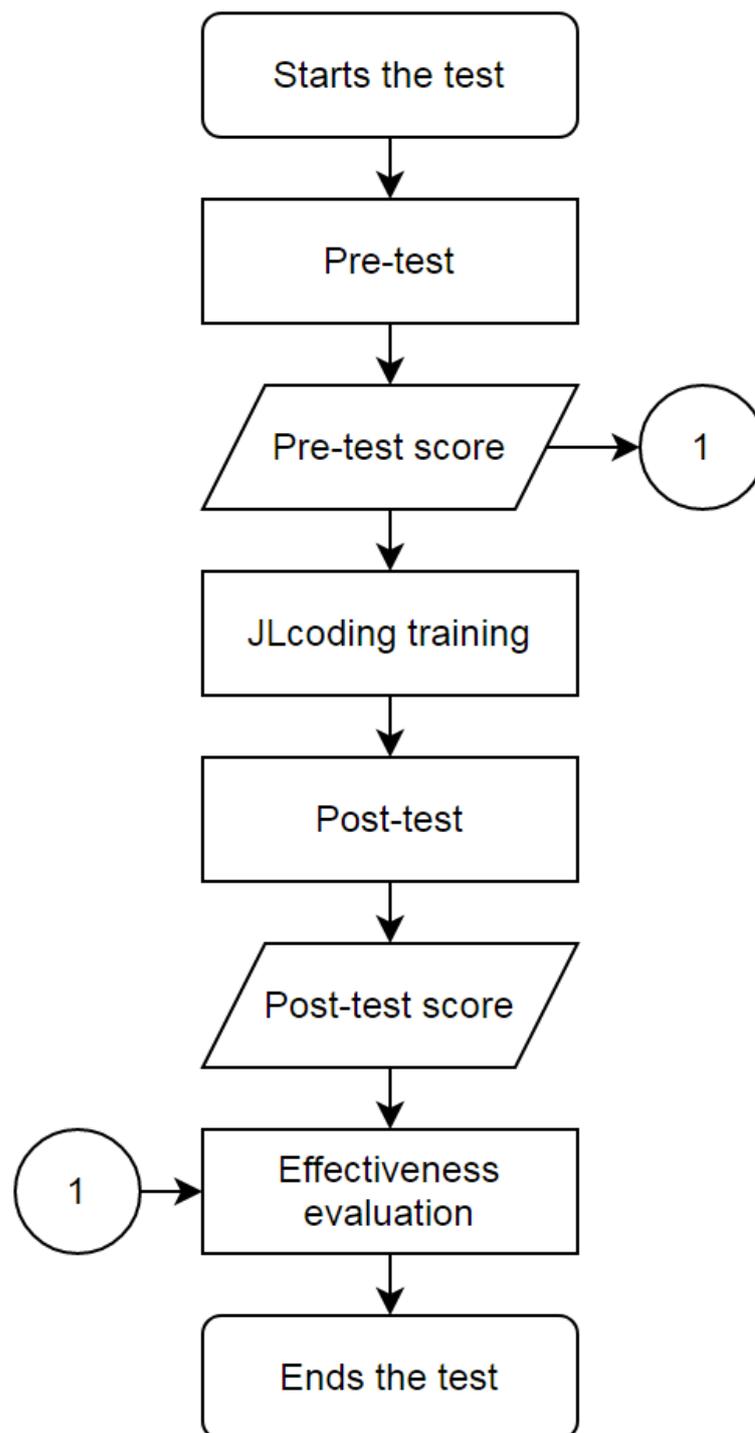


Figure 23. The experimental flowchart.

4.3. Survey Design

The first step in data collection is to use a questionnaire to determine whether the student has ever learned a programming language before. Figure 24 shows our preliminary questionnaire to collect information about students programming language backgrounds. This questionnaire is essentially needed for the student selection process and grouping in the early steps of the research. To compare the difference between pre-test and post-test, the basic information included the student's name, gender, past programming experience, and whether they were interested in JLcoding.

Questionnaire

1. Basic information

(1) Class: _____ Name: _____

(2) Gender: Male Female

(3) Which programming courses have you taken (multiple choice):

Scratch Code.org mBlock Python Other: _____

Never

(4) Are you interested in JLcoding courses?

interested non-interested

Figure 24. Preliminary questionnaire.

In order to test whether the student who learned JLcoding can fit the requirement of a 12-year national basic education syllabus, this study included a test to include all the basic knowledge that they should know. For example, data forms, variables, input/output, arithmetic operations, logical operations, selection structure, and repetition. Figure 25 shows the pre- and post-testing questions that were used in the experiments. The question consists of 10 multiple-choice questions and 4 filling questions, which represent some of the abilities that one must have when coding, such as name, variable, operator, if, loop, and so on. The second part includes the fill-in questions. The student needs to fill in the C++, Java, and Python program codes to answer the questions. The first question of part (2) in Figure 25c is used to test the “input/output” ability of the student. The student needs to imitate the code shown above to output their name on the screen. The second question in Figure 25d is used to test the arithmetic operations ability. The student needs to type the right code for computing the value of “a + b.” The third question is used to test the logical ability of the student. The student needs to know how to write the structure of “if” to obtain the true answer. The fourth question is used to test the repetition ability of the student. They need to know how to write the “For loop” to print the number from 1 to 5.

After the testing process, we provided a questionnaire to the students to determine the results of student’s confidence in the programming language that had been learned. There are four questions in the confidence questionnaire given in the form of a Likert scale with a range of 1–5. The four questions are:

1. Writing a computer program is easy.
2. I am good at writing computer programs.
3. I plan to continue programming after the class is over.
4. I want to take another computer programming.

Questionnaire

1. Basic information
 (1) Class : _____ Name : _____
 (2) Gender : Male Female
 (3) Which programming courses have you taken (multiple choice) :
Scratch Code.org mBlock Python Other: _____
Never
 (4) Are you interested in JLCoding course?
Interested Non-interested

(1) Choose the correct answer

() 1. To declare a bee object in the program, which naming method is more correct?

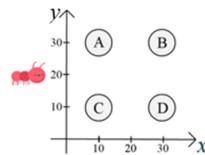
- (A) 1 蜜蜂
- (B) 蜜蜂 1 隻
- (C) 1bee
- (D) beel



() 2. When the system executes the following program, where will the ant move to?

```
ant.x = 10
ant.y = 30
```

- (A) A Location
- (B) B Location
- (C) C Location
- (D) D Location



() 3. The height of a giraffe is three times that of a penguin. If penguin is 150, how do you define the height of a giraffe?

- (A) giraffe.height = 150
- (B) giraffe.width = 150
- (C) giraffe.height = 450
- (D) giraffe.width = 450



() 4. The pig is too big to pass through the door. You must use the scale program to make the pig smaller. What number should you fill in the blank?
 pig.scaleX = □

```
pig.scaleY = □
```

- (A) 100
- (B) 10
- (C) 1
- (D) 0.1



() 5. Different colors in the program are represented by different codes. Red is #FF0000, green is #00FF00, and blue is #0000FF. When the system executes the following program, what color will the crystal ball turn into?
 ball.innerColor = #00FF00

- (A) red
- (B) green
- (C) blue
- (D) nchanged



() 6. Which program should I use to get the turtle close to the egg?

- (A) turtle.run()
- (B) turtle.right()
- (C) turtle.forward()
- (D) turtle.前進()



() 7. Which program should I use to let the cat eat the fish on the left?

- (A) cat.go("F", 800, 500)
- (B) cat.go("L", 600, 700)
- (C) cat.go("U", 400, 1100)
- (D) cat.go("D", 200, 1500)



() 8. If you want to copy a bomb, which program should you use?

- (A) get = new.bomb(1)
- (B) clone = g.clone(bomb)
- (C) bomb.clone(360, 1, 1500)
- (D) bomb.clone(5, 9, 30, 1500)



(a)

(b)

() 9. The boy calculates how much money is in the box. After the system executes the following program, how much money is in the box?

```
box.data = 800
box.data = box.data - 600
boy.say(box.data, 400, #000000, 200, 1000)
```

- (A) 200
- (B) 600
- (C) 800
- (D) 800



() 10. Continuing from the above question, the little boy wants to use the money in the box to go abroad. When the system executes the following program, will the little boy take a ship or a boat?

```
if (box.data >= 500)
    boy.moveToCenter(ship)
else
    boy.moveToCenter(boat)
end
```

- (A) ship
- (B) boat
- (C) swimming
- (D) not going abroad



(2) Fill in the correct answer in the blank.

1. Please enter the program in the blank to output your name.

cout << "Hello"; cout << "my"; cout << "name"; cout << "is"; □	System.out.print ("Hello"); System.out.print ("my"); System.out.print ("name"); System.out.print ("is"); □	print ("Hello") print ("my") print ("name") print ("is") □
--	--	--

2. Please use an addition program to make C equal to 3.

int a=1; int b=2; int c; □ cout << c;	int a=1; int b=2; int c; □ System.out.print (c);	a=1 b=2 □ print (c)
---	--	------------------------------

3. Please use IF syntax to input 10 divided by 2 equal 5 and output the "true" string.

(c)

(d)

□ { cout << "true"; } else { cout << "false"; }	□ { System.out.print ("true"); } else { System.out.print ("false"); }	□ print ("true") else: print ("false") }
--	--	--

4. Please use the For loop to output numbers 1 to 5.

□ { cout << i; }	□ { System.out.print (i); }	□ for i in sequences: print (i)
---------------------------	--------------------------------------	---------------------------------------

(3) Tick the level of consent according to how you feel.

Item	Strongly disagree 1	Disagree 2	Neutral 3	Agree 4	Strongly Agree 5
1. Writing a computer program is easy.	<input type="checkbox"/>				
2. I am good at writing computer programs.	<input type="checkbox"/>				
3. I plan to continue programming after the class is over.	<input type="checkbox"/>				
4. I want to take another computer programming.	<input type="checkbox"/>				

Figure 25. A sample JLCoding test: (a) multiple-choice questions of Named, Coordinate, Height and Width, Scale; (b) multiple-choice questions of Color Code, Go Ahead, Move, Clone; (c) multiple-choice questions of Variable and if/else; (d) filling questions and confidence investigation.

4.4. Data Processing and Analysis

The research process was divided into three stages: pre-test, course training, and post-test. All questionnaires are inquired in paper form and collected immediately after filling out. The results of the questionnaire were analyzed by SPSS Statistics 22 software (IBM, New York, NY, USA). As shown in Figure 26, the analysis method is as follows:

- Use descriptive statistics to illustrate the performance of students before and after the JLCoding course.
- Use the paired-sample *t*-test to illustrate the performance of students before and after the JLCoding course.
- Use the independent-sample *t*-test to show whether there is a significant difference in the progress scores of students of different genders in the JLCoding course.
- Use the independent-sample *t*-test to show whether there is a significant difference in the progress scores of students interested in JLCoding and those who are not interested in learning JLCoding courses.
- Use the independent-sample *t*-test to illustrate the degree of confidence of different genders and interests in students' programming abilities.

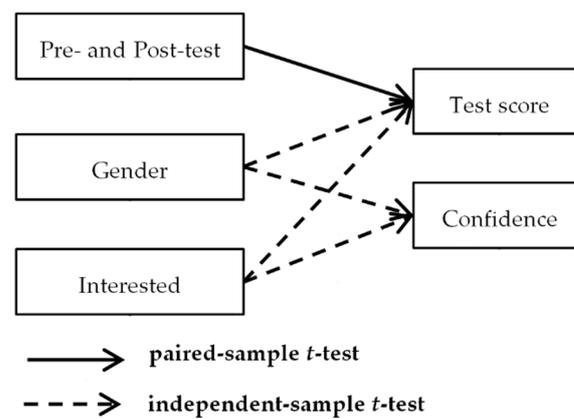


Figure 26. Research architecture diagram.

4.5. Descriptive Statistics

The experiment of this research was carried out from October 2020 to January 2021. Table 4 shows the descriptive statistics of the research participants. A total of 41 questionnaires were distributed, two participants dropped out in the process, four questionnaires were invalid, and a total of 35 participants completed the test.

Table 4. Descriptive statistics.

Attribute	Participants	Percentage
Male	18	51.4%
Female	17	48.6%
Scratch	34	97.1%
Code.org	7	20.0%
mBlock	5	14.3%
Python	2	5.7%
Interested in JLCoding	24	68.6%
Non-interested in JLCoding	11	31.4%

Among them, males made up 51.4% and females 48.6%. All the students had taken relevant programming courses, including Scratch. After the test, students were asked whether they were interested in JLCoding, and 68.6% of the students expressed interest.

4.5.1. JLCoding Test

From Table 5, it can be observed that the students performed well in the pre-test averages of Coordinate, Height and Width, Color Code, and Move, while the pre-test performances of Scale, Clone, Variable, and If/else were poor. We compared the performance of the students before and after teaching. Except for Question 8, the student's correct answers to other questions have increased.

Table 5. JLCoding test score comparison.

Question	Attribute	Pre-Test	Post-Test	Progress
Question 1	Named	6.3	9.4	+49.21%
Question 2	Coordinate	8.3	9.1	+9.64%
Question 3	Height and Width	8.6	9.7	+12.79%
Question 4	Scale	2.9	7.1	+144.83%
Question 5	Color Code	8.3	8.9	+7.23%
Question 6	Go Ahead	3.7	4.3	+16.22%
Question 7	Move	6.9	7.4	+7.25%
Question 8	Clone	1.7	1.1	−35.29%
Question 9	Variable	2.0	4.9	+145.00%
Question 10	If/else	4.3	6.0	+39.53%

Scale, Variable, and If/else are items with lower scores in the pre-test, and the average score growth in the post-test is greater than 30%. The average pre-test scores of Coordinate, Color Code, and Move are already very high, and it is not easy to have room for improvement, so the degree of improvement is small.

Table 6 shows the comparisons of the average scores of the pre-test and post-test. The post-test is 15.14, higher than the average score of the pre-test. SD is the standard deviation, SE is the standard error, and N is the total number of students. Tables 7 and 8 show the statistics of the number of scores before and after the test. The data of the number of participants and the percentage of the number of participants are counted in an interval of 10. Figure 27 shows that the score of 50 has the highest frequency in the pre-test, and that of 80 has the highest frequency in the post-test, which means that the score has significantly improved.

Table 6. JLCoding test total score comparison.

Test	N	Mean	SD	SE
pre-test	35	52.86	18.720	3.164
post-test	35	68.00	14.912	2.521

Table 7. Pre-test score.

Score	N	Percentage
0	1	2.7%
10	1	2.7%
30	2	5.4%
40	6	16.2%
50	9	24.3%
60	8	21.6%
70	4	10.8%
80	3	13.5%
90	1	2.7%

Table 8. Post-test score.

Score	N	Percentage
20	1	2.9%
30	1	2.9%
50	2	5.7%
60	9	25.7%
70	9	25.7%
80	11	31.4%
90	2	5.7%

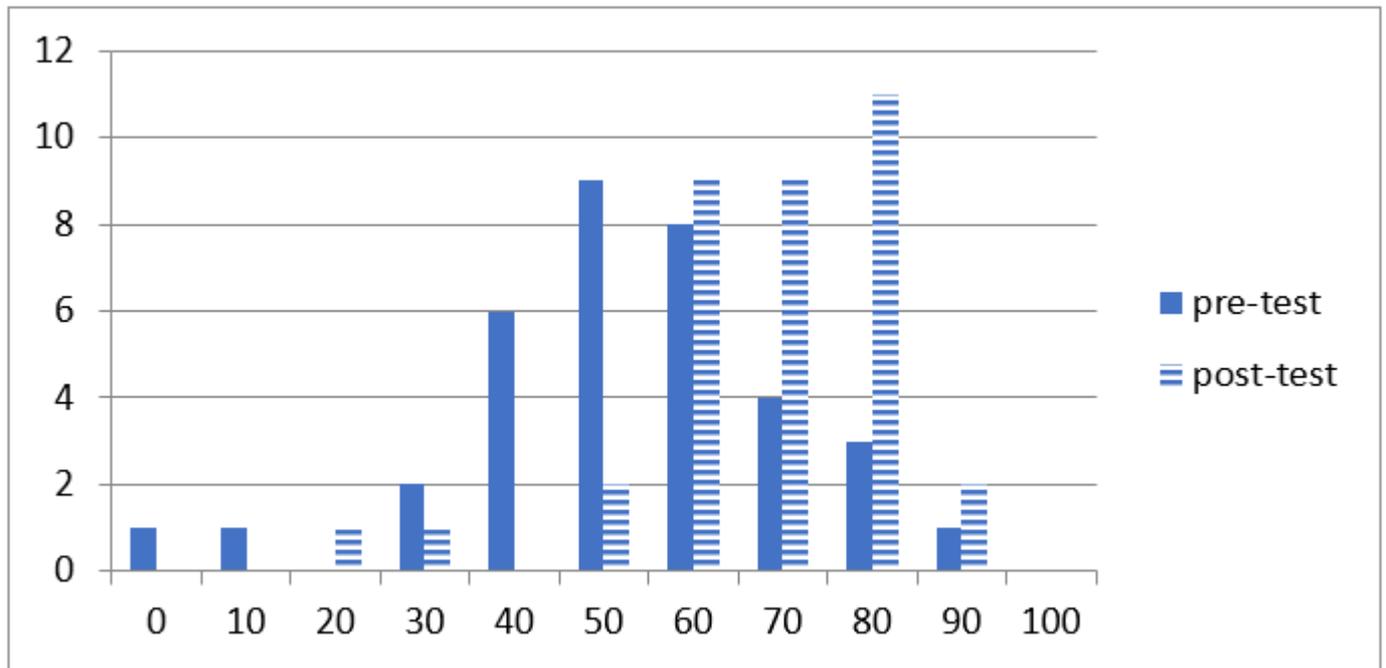


Figure 27. Histogram of pre-test and post-test scores.

This study divides the students into three different categories, high-level, middle-level, and low-level groups, according to the score of the pre-test. The top 25% of students are categorized under the high-level group, and the last 25% of students are categorized under the low-level group. The remaining students are categorized into the middle-level group. The three groups are used to calculate their progress. The performance after teaching and training, as well as the experimental results, shows that the average score of the low-level group improved by 25.56, the middle-level group improved by 17.65, and the high-level group improved by 0. The results are shown in Table 9.

Table 9. Statistics of low, middle, and high groups.

Category	N	Pre-Test Mean	Post-Test Mean	Progress Score
low-level	9	30	55.56	25.56
middle-level	17	53.53	71.18	17.65
high-level	9	74.44	74.44	0.00

The degree of improvement is related to the pre-teaching scores. Those with lower pre-test scores have more room for improvement. Conversely, those with higher pre-test scores have less room for improvement.

Furthermore, the study categorizes the students into male and female groups according to their gender for statistical analysis. The average gap between the pre-test with a different gender is 2.45. After training and learning, both groups improved by more than 10. The gap

in the post-testing scores between the two groups is only 0.46. The degree of the progress score of the male group is higher than that of the female, as shown in Table 10.

Table 10. Statistics of male and female groups.

Category	N	Pre-Test Mean	Post-Test Mean	Progress Score
male	18	51.67	67.78	16.11
female	17	54.12	68.24	14.12

The results in Table 11 show our group comparisons based on students' interest in JLCoding. The gap in the pre-test score between the two groups is 3.78. After training and learning, the interested group improved by more than 10, while the non-interested group improved by 10. The gap in the post-test scores between the two groups is 3.72. Furthermore, the degree of improvement in the interested group is higher than that in the non-interested group.

Table 11. The comparisons are based on students' interest in JLCoding.

Category	N	Pre-Test Mean	Post-Test Mean	Progress Score
interested	24	51.67	69.17	17.50
non-interested	11	55.45	65.45	10

4.5.2. Text-Based Programming Quiz

In the text-based programming quiz, four types of fill questions were used to test whether the students can transform from block-based to text-based programming. The questions included the output string, calculation if/else, and loop. For example, Figure 28 shows the questions in the quiz: (a) The student needs to follow the example to fill in how to write their name in C++, Java, and Python, (b) how to set the variable c to be the result of a + b, (c) judge whether $10/2 = 5$ is true or false, and (d) set up the "for" loop.

1. Please enter the program in the blank to output your name.

cout << "Hello"; cout << "my"; cout << "name"; cout << "is"; <input type="text"/>	System.out.print ("Hello"); System.out.print ("my"); System.out.print ("name"); System.out.print ("is"); <input type="text"/>	print ("Hello") print ("my") print ("name") print ("is") <input type="text"/>
---	---	---

(a)

2. Please use an addition program to make C equal to 3.

int a=1; int b=2; int c; <input type="text"/> cout << c;	int a=1; int b=2; int c; <input type="text"/> System.out.print (c);	a=1 b=2 <input type="text"/> print (c)
--	---	---

(b)

3. Please use IF syntax to input 10 divided by 2 equal 5 and output the "true" string.

<input type="text"/> { cout << "true"; } else { cout << "false"; }	<input type="text"/> { System.out.print ("true "); } else { System.out.print ("false "); }	<input type="text"/> print ("true ") else: print ("false ")
---	---	--

(c)

4. Please use the For loop to output numbers 1 to 5.

<input type="text"/> { cout << i; }	<input type="text"/> { System.out.print (i); }	<input type="text"/> for i in sequences: print (i)
--	---	--

(d)

Figure 28. Text-based programming quiz: (a) the output string question of the quiz; (b) the calculation question of the quiz; (c) the if/else question of the quiz; (d) the loop question of the quiz.

Table 12 shows the efficiency of text-based program testing. For the first question, the percentages of the students who can answer correctly for C++, Java, and Python are 42%, 44%, and 80%, respectively. The percentages for the second question are 36%, 30%, and 30%; for the third question are 24%, 20%, and 20%. No one can answer the fourth question. Most of the students can answer the first question. Most of the students can answer correctly in Python format.

Table 12. Test of text-based programming.

Quiz	Category	Correct		Wrong		Give Up	
1. Output String	C++	15	(42%)	14	(40%)	6	(16%)
	Java	16	(44%)	13	(36%)	6	(16%)
	Python	28	(80%)	1	(2%)	6	(16%)
2. Calculate	C++	13	(36%)	15	(42%)	7	(20%)
	Java	11	(30%)	14	(40%)	10	(28%)
	Python	11	(30%)	13	(36%)	11	(30%)
3. If/else	C++	9	(24%)	4	(10%)	22	(62%)
	Java	7	(20%)	5	(14%)	23	(64%)
	Python	7	(20%)	4	(10%)	24	(68%)
4. Loop	C++	0	(0%)	8	(22%)	27	(76%)
	Java	0	(0%)	6	(16%)	29	(82%)
	Python	0	(0%)	6	(16%)	29	(82%)

Most students are willing to try to answer the question without formally learning text-based programming. In the calculation question, half of the students can choose the correct answers; however, for the If/else and loop question, most of them had given up answering because, in the experiment, the teaching material only included Name, Coordinate, Height and Width, Scale, Color Code, Go Ahead, Move, Clone, Variable, and If/else. The For loop course was not included in this period. Furthermore, the If/else course was the final course—students did not have enough time to practice.

From the experiments, we can see that the students have some ability to transform from block-based programming to text-based programming for the part that they have learned in JLCoding.

4.5.3. Programming Confidence Questionnaire

This study investigated whether students of JLCoding had confidence in their ability to write computer programs and were asked whether they wanted to continue programming. Table 13 shows the results of students' confidence in programming. The value of C2 is lower than 3, which means that the student's self-confidence in computer programming decreased; however, the values of the other questions are high. They believe that writing computer programs is easy and they want to write computer programs after class and enroll in other programming courses, which shows that students enjoy the programming experience.

Table 13. Programming of confidence.

No.	Question	Mean	SD
C1	Writing computer programs is easy.	3.03	1.071
C2	I am good at writing computer programs.	2.97	1.200
C3	I plan to continue programming after the class is over.	3.11	1.132
C4	I want to take another computer programming course.	3.29	1.073

4.6. Paired Samples *t*-Test

In terms of statistical methods, the *t*-test is used to analyze whether there is a difference in the sample averages of the two groups, and when the two sets of samples are related, the paired samples *t*-test can be used. This study compares paired data through this verification method. The study analyzes the difference between the pre-test and post-test scores. The hypothesis of the *t*-test is the Null and Opposite hypothesis:

- Null hypothesis (H0): There is no significant difference between the pre-test and post-test scores.

- Opposite hypothesis (H1): There is a significant difference between the pre-test and post-test scores.

The analysis results are shown in Table 14, where t is the test quantity. The t -value of the test statistic method subtracting post-test from the pre-test score. Usually, the post-test score is greater than the pre-test score, such that the t -value is a negative number. Both the upper and lower bounds of the confidence interval are negative numbers and the range does not contain 0. This shows that under 95% probability, the two will not be the same. A significant p is less than 0.000, which means that the pre-test and post-test have the same probability of less than 0.000. The significance p is less than 0.05, which means rejecting the null hypothesis H_0 .

Table 14. Paired-sample t -test of pre-test and post-test.

Test	Mean	SD	SEM	Paired Differences		t	df	p
				95% Confidence Interval of the Difference				
				Lower	Upper			
Pre-test–Post-test	−15.143	19.154	3.238	−21.723	−8.563	−4.677	34	0.000

The opposite hypothesis is established, and there is a significant difference between the pre-test (before the course) and the post-test scores (after the course); therefore, we saw that the student had a significant difference in grades after studying JLCoding. In other words, students can improve their grades significantly after using the JLCoding course.

4.7. Independent Samples t -Test

If the two samples are independent and do not affect each other, the independent samples t -test is required. The samples must have the characteristics of an independent event, i.e., the two samples will not affect each other. This research uses gender to know if the students are interested in the system to perform an independent sample t -test. First, we analyze whether students of different genders have significant differences in their progress scores after using this system. The hypothesis of the t -test is:

- Null hypothesis (H_0): There is no significant difference between the two groups of students.
- Opposite hypothesis (H_1): There are significant differences between the two groups of students.

The analysis results are shown in Tables 15 and 16. There was a non-significant difference in the progress scores for males ($M = 16.11$, $SD = 20.62$) and females ($M = 14.12$, $SD = 18.05$). The p for the gender score is 0.763, which is greater than 0.05. The value shows that the standard did not reach a significant degree. The null hypothesis is accepted. Males and females have no significant difference in their progress after using this system; however, regardless of whether males and females use this system to study, their progress is independent of their gender.

Table 15. Statistics of males and females.

Gender	N	Mean	SD	SE
male	18	16.11	20.62	4.86
female	17	14.12	18.05	4.38

Table 16. Independent sample *t*-test of males and females.

Gender scores	<i>t</i> -Test for Equality of Means						
	<i>t</i>	df	<i>p</i>	Mean Difference	SE Difference	95% Confidence Interval of the Difference	
						Lower	Upper
	0.304	33	0.763	1.99346	6.56627	−11.36571	15.35263

The second analysis was analyzed to know whether the score has a relationship with their interest in JLCoding and if there is a significant difference in progress after using this system.

- Null hypothesis (H0): There is no significant difference between the two groups of students.
- Opposite hypothesis (H1): There are significant differences between the two groups of students.

The analysis results are shown in Tables 17 and 18. There was a non-significant difference in the progress scores for the student with interest ($M = 17.50$, $SD = 16.75$) and non-interested ($M = 10.00$, $SD = 23.66$). The *p* for the interest score is 0.289, which is larger than 0.05. The value shows that the standard did not reach a significant degree. The null hypothesis is accepted. Nevertheless, regardless of whether or not they are interested in using this system to study, their progress is independent of their interest.

Table 17. Statistics of interest and non-interest.

Interested	N	Mean	SD	SE
yes	24	17.50	16.75	3.42
no	11	10.00	23.66	7.14

Table 18. Independent sample *t*-test of interest and non-interest.

Gender scores	<i>t</i> -Test for Equality of Means						
	<i>t</i>	df	<i>p</i>	Mean Difference	SE Difference	95% Confidence Interval of the Difference	
						Lower	Upper
	1.078	33	0.289	7.50000	6.95775	−6.65564	21.65564

Finally, we used an independent-sample *t*-test to compare whether there are differences in gender and interest in learning JLCoding. Tables 17 and 18 show the results of the *t*-test. In Table 19, C3 (I plan to continue programming after the class is over) is less than 0.05. In Table 20, C2 (I am good at writing computer programs) and C3 are less than 0.05. This shows that gender and interest have a significant impact on these two projects. In other words, males will be more willing to learn new things in programming than females. Consequently, students with programming interests are more likely to think they are good at programming and maintain the willingness to learn.

Table 19. Independent samples *t*-test of gender.

No.	Gender				<i>t</i> -Test Results for Interested	
	Male		Female		<i>t</i>	<i>p</i>
	Mean	SD	Mean	SD		
C1	3.33	1.283	2.71	0.686	1.817	0.081
C2	3.33	1.283	2.56	1.004	1.905	0.065
C3	3.50	1.098	2.71	1.047	2.187	0.036
C4	3.56	1.042	3.00	1.061	1.563	0.128

Table 20. Independent samples *t*-test of interested.

No.	Interested				<i>t</i> -Test Results for Interested	
	Yes		No		<i>t</i>	<i>p</i>
	Mean	SD	Mean	SD		
C1	3.25	0.944	2.55	1.214	1.873	0.070
C2	3.29	0.999	2.27	1.348	2.506	0.017
C3	3.42	0.974	2.45	1.214	2.510	0.017
C4	3.50	0.978	2.82	1.168	1.802	0.081

5. Conclusions

In this paper, we proposed the JLCoding as a tool for early programming learning. We found that there are no difficulties in learning text-based programming using JLCoding. Students can understand the basics of programming through the JLCoding course and expand to more complex programming concepts.

It can be seen from the experimental analysis results that this system can indeed make progress in programming. More significant progress can be made, especially for those with lower scores in the pre-test. Students can improve their scores through a series of teaching materials, irrespective of their gender or interest in the system or not.

However, our findings show that males are more willing to learn the program, and students who have more interest in the system are more confident in programming. Only C2 (I am good at writing computer programs) in the confidence questionnaire is lower than the rest of the items in the questionnaire, which may be related to the reasons mentioned by Colleen M. Lewis, stating that students do not think they have fully learned all the functions in JLCoding [13], which is in line with the data in this study. Students performed poorly on variables, conditional expressions, and loops in the test. The reason could be that these functions are taught at the end of the course, resulting in insufficient time for students to practice. If students can have more time to practice, they could perform better.

Because the students in this study are limited to secondary school bilingual colleges, and the seven-week course is too long to obtain many samples in a short period. Students in this study had never learned Scratch. The experimental results cannot show the learning effectiveness of the students who have never studied programming. Hence, the study will test JLCoding, the students who have never learned any program before to compare the teaching performance.

In the future, the research will extend the testing groups to elementary and high schools to obtain more data. The programming of basic grammar is currently the main teaching objective of this system. Nevertheless, when the level of students has improved, more complex programming courses, such as game development, Internet of Things, and artificial intelligence, will be added to the system. If more participants can be recruited for research in the future, the experimental group can be compared with the control group to understand the differences between the students using other programming education software. Additionally, the views of the students' parents or teachers on the software will be collected to further understand the complete satisfaction and effectiveness of its use.

Author Contributions: Conceptualization, W.-Y.L.; methodology, T.-C.L.; software, W.-Y.L.; validation, T.-C.L.; writing—original draft preparation, W.-Y.L.; writing—review and editing, T.-C.L.; All authors have read and agreed to the published version of the manuscript.

Funding: Ministry of Science and Technology (MOST), Taiwan, Republic of China, under the Grant MOST 109-2221-E-324 -025 -MY3.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Effenberger, T.; Pelánek, R. Towards making block-based programming activities adaptive. In Proceedings of the 5th annual ACM Conference Learned Scale, London, UK, 26–28 June 2018; pp. 1–4.
2. Israel, M.; Pearson, J.N.; Tapia, T.; Wherfel, Q.M.; Reese, G. Supporting all learners in school-wide computational thinking: A cross-case qualitative analysis. *Comput. Educ.* **2015**, *82*, 263–279. [\[CrossRef\]](#)
3. Kucuk, S.; Sisman, B. Behavioral patterns of elementary students and teachers in one-to-one robotics instruction. *Comput. Educ.* **2017**, *111*, 31–43. [\[CrossRef\]](#)
4. Manches, A.; Plowman, L. Computing education in children’s early years: A call for debate. *Br. J. Educ. Technol.* **2017**, *48*, 191–201. [\[CrossRef\]](#)
5. Mannila, L.; Dagiene, V.; Demo, B.; Grgurina, N.; Mirolo, C.; Rolandsson, L.; Settle, A. Computational thinking in K-9 Education. In Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference (ITiCSE-WGR ’14), Uppsala, Sweden, 23–25 June 2014; pp. 1–29.
6. Webb, M.; Davis, N.; Bell, T.; Katz, Y.J.; Reynolds, N.; Chambers, D.P.; Sysło, M.M. Computer science in K-12 school curricula of the 21st century: Why, what and when? *Educ. Inf. Technol.* **2017**, *22*, 445–468. [\[CrossRef\]](#)
7. Wong, G.; Jiang, S.; Kong, R. Computational thinking and multifaceted skills: A qualitative study in primary schools. In *Teaching Computational Thinking in Primary Education*; IGI Global: Hershey, PA, USA, 2018; pp. 78–101.
8. Grover, S.; Basu, S. Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and Boolean logic. In Proceedings of the ACM SIGCSE Technical Symposium on Computer Science Education, Seattle WA, USA, 1–8 March 2017; pp. 267–272.
9. Lewis, C.; Esper, S.; Bhattacharyya, V.; Fa-Kaji, N.; Dominguez, N.; Schlesinger, A. Children’s perceptions of what counts as a programming language. *J. Comput. Sci. Coll.* **2014**, *29*, 123–133.
10. Weintrop, D.; Wilensky, U. Comparing block-based and text-based programming in high school computer science classrooms. *ACM Trans. Comput. Educ.* **2017**, *18*, 1–25. [\[CrossRef\]](#)
11. Grover, S.; Pea, R. Computational thinking in K-12: A review of the state of the field. *Educ. Res.* **2013**, *42*, 38–43. [\[CrossRef\]](#)
12. Wing, J.M. Computational thinking. *Commun. ACM* **2006**, *49*, 33–35. [\[CrossRef\]](#)
13. Lewis, C.M. How programming environment shapes perception, learning and goals: Logo vs. scratch. In Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE ’10), Milwaukee, WI, USA, 10–13 March 2010; pp. 346–350.
14. Gretter, S.; Yadav, A. Computational thinking and media & information literacy: An integrated approach to teaching twenty-first century skills. *TechTrends* **2016**, *60*, 510–516.
15. Hagge, J. Scratching beyond the surface of literacy: Programming for early adolescent gifted students. *Gift. Child Today* **2017**, *40*, 154–162. [\[CrossRef\]](#)
16. Kalantzis, M.; Cope, B. New learning: A charter for change in education 1. *Crit. Stud. Educ.* **2012**, *53*, 83–94. [\[CrossRef\]](#)
17. Benander, A.; Benander, B.; Sang, J. Factors related to the difficulty of learning to program in Java—An empirical study of non-novice programmers. *Inf. Softw. Technol.* **2004**, *46*, 99–107. [\[CrossRef\]](#)
18. Gomes, A.; Mendes, A.J. Learning to program—difficulties and solutions. In Proceedings of the International Conference on Engineering Education-ICEE, Coimbra, Portugal, 3–7 September 2007; pp. 283–287.
19. Rahmat, M.; Shahrani, S.; Latih, R.; Yatim, N.F.M.; Zainal, N.F.A.; Rahman, R.A. Major problems in basic programming that influence student performance. *Procedia-Soc. Behav. Sci.* **2012**, *59*, 287–296. [\[CrossRef\]](#)
20. Watson, C.; Li, F.W.B. Failure rates in introductory programming revisited. In Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education (ITiCSE ’14), Uppsala, Sweden, 21–25 June 2014; pp. 39–44.
21. Smith, R.; Rixner, S. The error landscape: Characterizing the mistakes of novice programmers. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE ’19), Minneapolis, MN, USA, 27 February–2 March 2019; pp. 538–544.
22. Combéfis, S.; Beresnevičius, G.; Dagiene, V. Learning programming through games and contests: Overview, characterisation and discussion. *Olymp. Inform.* **2016**, *10*, 39–60. [\[CrossRef\]](#)

23. Qian, Y.; Lehman, J. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Trans. Comput. Educ.* **2017**, *18*, 1–24. [CrossRef]
24. Seralidou, E.; Douligieris, C. Learning programming by creating games through the use of structured activities in secondary education in Greece. *Educ. Inf. Technol.* **2021**, *26*, 859–898. [CrossRef]
25. Funke, A.; Geldreich, K.; Hubwieser, P. Analysis of Scratch projects of an introductory programming course for primary school students. In Proceedings of the 2017 IEEE Global Engineering Education Conference (EDUCON), Athens, Greece, 25–28 April 2017; pp. 1229–1236.
26. Park, Y.; Shin, Y. Comparing the effectiveness of scratch and app Inventor with Regard to Learning Computational Thinking Concepts. *Electronics* **2019**, *8*, 1269. [CrossRef]
27. Price, T.W.; Barnes, T. Comparing textual and block interfaces in a novice programming environment. In Proceedings of the Eleventh Annual International Conference on International Computing Education Research (ICER '15), Omaha, NE, USA, 9–13 July 2015; pp. 91–99.
28. Parsons, D.; Haden, P. Programming osmosis: Knowledge transfer from imperative to visual programming environments. In Proceedings of the 20th Annual Conference of the National Advisory Committee on Computing Qualifications (NACCQ2007), Nelson, New Zealand; Mann, S., Bridgeman, N., Eds.; pp. 209–215. Available online: https://www.researchgate.net/publication/228525906_Programming_osmosis_Knowledge_transfer_from_imperative_to_visual_programming_environments (accessed on 24 May 2022).
29. Moors, L.; Luxton-Reilly, A.; Denny, P. Transitioning from block-based to text-based programming languages. In Proceedings of the International Conference on Learning and Teaching in Computing and Engineering (LATICE), Auckland, New Zealand, 19–22 April 2018; pp. 57–64.
30. Symmetry of Language. Available online: <https://wiki.c2.com/?SymmetryOfLanguage> (accessed on 14 May 2022).
31. Luxton-Reilly, A.; Albluwi, I.; Becker, B.A.; Giannakos, M.; Kumar, A.N.; Ott, L.; Paterson, J.; Scott, M.J.; Sheard, J.; Szabo, C. Introductory programming: A systematic literature review. In Proceedings of the Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018 Companion), Larnaca, Cyprus, 2–4 July 2018; pp. 55–106.
32. Beaubouef, T.; Mason, J. Why the high attrition rate for computer science students: Some thoughts and observations. *ACM SIGCSE Bull.* **2005**, *37*, 103–106. [CrossRef]
33. Marwan, S.; Lytle, N.; Williams, J.J.; Price, T. The impact of adding textual explanations to next-step hints in a novice programming environment. In Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '19), Aberdeen, UK, 15–17 July 2019; pp. 520–526.
34. Wiggins, J.B.; Fahid, F.M.; Emerson, A.; Hinckle, M.; Smith, A.; Boyer, K.E.; Mott, B.; Wiebe, E.; Lester, J. Exploring novice programmers' hint requests in an intelligent block-based coding environment. In Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21), Virtual Event, USA, 13–20 March 2021; pp. 52–58.
35. Bennedsen, J.; Caspersen, M.E. Failure rates in introductory programming: 12 years later. *ACM Inroads* **2019**, *10*, 30–36. [CrossRef]
36. Vihavainen, A.; Airaksinen, J.; Watson, C. A systematic review of approaches for teaching introductory programming and their influence on success. In Proceedings of the Tenth Annual Conference on International Computing Education Research (ICER '14), Glasgow, UK, 11–13 August 2014; pp. 19–26.
37. Giannakos, M.N.; Pappas, I.O.; Jaccheri, L.; Sampson, D.G. Understanding student retention in computer science education: The role of environment, gains, barriers and usefulness. *Educ. Inf. Technol.* **2017**, *22*, 2365–2382. [CrossRef]
38. Kinnunen, P.; Malmi, L. Why students drop out CS1 course? In Proceedings of the Second International Workshop on Computing Education Research (ICER '06), Canterbury, UK, 9–10 September 2006; pp. 97–108.
39. Petersen, A.; Craig, M.; Campbell, J.; Tafliovich, A. Revisiting why students drop CS1. In Proceedings of the 16th Koli Calling International Conference on Computing Education Research (Koli Calling '16), Koli, Finland, 24–27 November 2016; pp. 71–80.
40. Techapalokul, P.; Tilevich, E. Understanding recurring quality problems and their impact on code sharing in block-based software. In Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Raleigh, NC, USA, 11–14 October 2017; pp. 43–51.
41. Weintrop, D. Block-based programming in computer science education. *Commun. ACM* **2019**, *62*, 22–25. [CrossRef]
42. Xu, Z.; Ritzhaupt, A.D.; Tian, F.; Umaphathy, K. Block-based versus text-based programming environments on novice student learning outcomes: A meta-analysis study. *Comput. Sci. Educ.* **2019**, *29*, 177–204. [CrossRef]
43. Bau, D.; Gray, J.; Kelleher, C.; Sheldon, J.; Turbak, F. Learnable programming: Blocks and beyond. *Commun. ACM* **2017**, *60*, 72–80. [CrossRef]
44. Xie, B.; Abelson, H. Skill progression in MIT app inventor. In Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Cambridge, UK, 4–8 September 2016; pp. 213–217.
45. Weintrop, D.; Wilensky, U. Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs. In Proceedings of the Eleventh Annual International Conference on International Computing Education Research (ICER '15), Omaha, NE, USA, 9–13 July 2015; pp. 101–110.
46. Armoni, M.; Gal-Ezer, J. Early computing education: Why? what? when? who? *ACM Inroads* **2014**, *5*, 54–59. [CrossRef]
47. Malik, S.I.; Mathew, R.; Al-Nuaimi, R.; Al-Sideiri, A.; Coldwell-Neilson, J. Learning problem solving skills: Comparison of e-learning and M-learning in an introductory programming course. *Educ. Inf. Technol.* **2019**, *24*, 2779–2796. [CrossRef]

48. Kazimoglu, C.; Kiernan, M.; Bacon, L.; Mackinnon, L. A serious game for developing computational thinking and learning introductory computer programming. *Procedia-Soc. Behav. Sci.* **2012**, *47*, 1991–1999. [CrossRef]
49. Zapušek, M.; Rugelj, J. Learning programming with serious games. *EAI Endorsed Trans. Game-Based Learn.* **2013**, *1*, e6. [CrossRef]
50. Du Boulay, B. Some difficulties of learning to program. *J. Educ. Comput. Res.* **1986**, *2*, 57–73. [CrossRef]
51. Ebrahimi, A. Novice programmer errors: Language constructs and plan composition. *Int. J. Hum.-Comput. Stud.* **1994**, *41*, 457–480. [CrossRef]
52. 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]
53. Denny, P.; Luxton-Reilly, A.; Tempero, E.; Hendrickx, J. Understanding the syntax barrier for novices. In Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education, Darmstadt, Germany, 27–29 June 2011; pp. 208–212.
54. Kelleher, C.; Pausch, R. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv.* **2005**, *37*, 83–137. [CrossRef]
55. Kölling, M.; Brown, N.C.C.; Altadmri, A. Frame-based editing: Easing the transition from blocks to text-based programming. In Proceedings of the Workshop in Primary and Secondary Computing Education (WiPSCE '15), London, UK, 9–11 November 2015; pp. 29–38.
56. Moskal, B.; Lurie, D.; Cooper, S. Evaluating the effectiveness of a new instructional approach. In Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '04), Norfolk, VA, USA, 3–7 March 2004; Volume 36, pp. 75–79.
57. Bishop-Clark, C.; Courte, J.; Howard, E.V. Programming in pairs with alice to improve confidence, enjoyment, and achievement. *J. Educ. Comput. Res.* **2006**, *34*, 213–228. [CrossRef]
58. Heintz, F.; Mannila, L.; Färnqvist, T. A review of models for introducing computational thinking, computer science and computing in K-12 education. In Proceedings of the 2016 IEEE Frontiers in Education Conference (FIE), Erie, PA, USA, 12–15 October 2016; pp. 1–9.
59. Passey, D. Computer science (CS) in the compulsory education curriculum: Implications for future research. *Educ. Inf. Technol.* **2017**, *22*, 421–443. [CrossRef]
60. Rose, S.P.; Habgood, M.P.J.; Jay, T. Designing a programming game to improve children's procedural abstraction skills in scratch. *J. Educ. Comput. Res.* **2020**, *58*, 1372–1411. [CrossRef]
61. Scratch Team 2020 Scratch Statistics. Available online: <https://scratch.mit.edu/statistics/> (accessed on 2 May 2021).
62. Rich, P.J.; Browning, S.F.; Perkins, M.; Shoop, T.; Yoshikawa, E.; Belikov, O.M. Coding in K-8: International trends in teaching elementary/primary computing. *TechTrends* **2019**, *63*, 311–329. [CrossRef]
63. Resnick, M.; Maloney, J.; Monroy-Hernández, A.; Rusk, N.; Eastmond, E.; Brennan, K.; Millner, A.; Rosenbaum, E.; Silver, J.; Silverman, B.; et al. Scratch: Programming for all. *Commun. ACM* **2009**, *52*, 60–67. [CrossRef]
64. Perkel, J.M.J. Julia: Come for the syntax, stay for the speed. *Nature* **2019**, *572*, 141–142. [CrossRef]
65. Bezanson, J.; Edelman, A.; Karpinski, S.; Shah, V.B. Julia: A fresh approach to numerical computing. *SIAM Rev.* **2017**, *59*, 65–98. [CrossRef]
66. Koolen, T.; Deits, R. Julia for robotics: Simulation and real-time control in a high-level programming language. In Proceedings of the International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 604–611.
67. Computer Programming of Khan Academy. Available online: <https://www.khanacademy.org/computing/computer-programming> (accessed on 24 June 2022).
68. Altadmri, A.; Kölling, M.; Brown, N.C.C. The Cost of Syntax and How to Avoid It: Text versus Frame-Based Editing. In Proceedings of the 40th Annual Computer Software and Applications Conference (COMPSAC), Atlanta, GA, USA, 10–14 June 2016; pp. 748–753.