electronics

MDPI

Article

Novel Scratch Programming Blocks for Web Scraping

Youngki Park [1] and Youhyun Shin [2,*]

[1] Department of Computer Education, Chuncheon National University of Education, Chuncheon 24328, Korea
[2] Department of Computer Science and Engineering, Incheon National University, Incheon 22012, Korea
* Correspondence: yhshin@inu.ac.kr

Abstract: Although Scratch is the most widely used block-based educational programming language, it is not easy for students to create various types of Scratch programs based on real-life data because it does not provide web scraping capabilities. In this paper, we present novel Scratch blocks for web scraping. Using these blocks, students can not only scrape the contents of HTML elements in a web page by using CSS selectors but also automate their keyboard and mouse in a number of ways, such as by using XPaths, the coordinates of the mouse, input strings, keys, or hot keys. We also present file access blocks that allow students to easily store and retrieve the scraped data in the form of key–value pairs. We conducted two lectures for a total of 15 primary/secondary school (K-12) teachers, allowing them to make ten web scraping example applications. As a result of a survey of the teachers, the proposed web scraping blocks achieved high scores for all evaluation measures.

Keywords: web scraping education; Scratch; Tooee; block-based programming languages; natural language processing

Citation: Park, Y.; Shin, Y. Novel Scratch Programming Blocks for Web Scraping. *Electronics* **2022**, *11*, 2584. https://doi.org/10.3390/electronics11162584

Academic Editor: George Angelos Papadopoulos

Received: 12 June 2022
Accepted: 16 August 2022
Published: 18 August 2022

1. Introduction

Web scraping is an important task for implementing practical applications, because many real-world problems can be solved by scraping and exploiting real-life data on the web. For example, in order to identify the most common job requirements in the field of medical informatics, the job advertisements from a job portal can be scraped [1]. As another example, in order to create a movie ranking program, it is necessary to scrape relevant information from movie rating websites such as IMDb. In addition, there are numerous other programs that require web scraping, such as the COVID-19 dashboard program or the weather forecasting program introduced in [2].

These applications can easily be implemented using text-based programming languages such as C++, Java, Python, or JavaScript. On the other hand, it is very hard (or impossible) to implement these applications using block-based programming languages such as Scratch [3,4] because these languages do not provide web scraping capabilities. For example, if primary school students who only know how to use Scratch want to create a movie ranking program, they must manually scrape the movie data from a movie website and hard-code the data into their program. At the time of writing this paper, when using block-based programming languages students find it difficult not only to create programs based on real-life data but also to fully achieve data-related computer science learning objectives (e.g., 1A-DA-05, 1A-DA-06, 1B-DA-06, and 2-DA-08 of the ACM CSTA K-12 Computer Science Standards [5]).

In this paper, we present novel Scratch programming blocks for web scraping in order to address this problem. With these blocks, students can easily perform web scraping using the block-based programming language. The main contributions of this paper are as follows:

- We describe the limitations of existing block-based programming languages in web scraping. After we briefly introduce block-based programming languages, we describe their web-scraping-related blocks (Section 2).

- We present novel Scratch blocks for web scraping. Using these blocks, students can not only scrape the contents of HTML elements in a web page by using CSS selectors but also automate their keyboard and mouse in a number of ways, such as by using XPaths, the coordinates of the mouse, input strings, keys, or hot keys. We also present file access blocks that can easily store and retrieve the scraped data in the form of key–value pairs (Section 3).
- We conducted two lectures for a total of 15 primary school teachers, allowing them to create ten web scraping example applications. We discuss the effectiveness of these blocks by analyzing the teachers' survey responses (Section 4).

The main research questions of this paper are as follows:

Research Question 1. Can primary/secondary school students (K-12 students) easily create practical programs using our web scraping blocks?

Research Question 2. Can primary/secondary school students improve their programming and data literacy skills by implementing web scraping programs using our blocks?

2. Related Work

Many students use block-based programming languages because of the advantage that the students can easily program by dragging and dropping programming blocks. Two of the most popular block-based programming languages are Scratch [3,4] and App Inventor [6,7]. As of May 2022, there were about 91,000,000 and 14,900,000 users registered on the Scratch and App Inventor websites, respectively [8,9]. These programming languages are mainly used for fostering computational thinking skills [10–19], which can be defined as "thinking like a computer scientist [20–22]. The block-based programming languages provide the basic building blocks of programming (such as loops, conditionals, functions, variables, and lists) so that young students can be used to create practical applications.

Block-based programming languages (especially Scratch) provide only a limited number of blocks, to make programming easier for students. This is a great advantage for students who are new to programming, but for students who are familiar with programming it can be an obstacle to creating their own programs. For example, students cannot use Scratch to access files on their own computers, nor can they scrape the contents of Google web pages.

There have been many approaches to addressing this problem, and the most common way is to provide additional blocks. For example, as shown on the left of Figure 1, *Cognimates* [23] provides blocks for scraping Twitter data. As another example, as shown on the right of Figure 1, *Machine Learning for Kids* [24] provides blocks for scraping Wikipedia data and blocks for scraping Twitter data. Obviously, these blocks are designed to be intuitive and easy for students to use. However, these approaches still have a limitation in that students cannot use these blocks to scrape data from websites other than Twitter or Wikipedia.

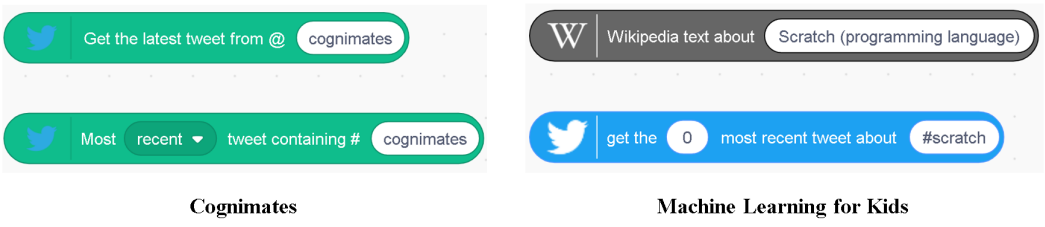

Figure 1. The existing blocks that can scrape data from Twitter or Wikipedia provided by Cognimates (on the **left**) and Machine Learning for Kids (on the **right**).

The *Entry* programming environment (or language) [25] supports web scraping in a slightly different way. It provides blocks for students to access spreadsheets in the cloud. In other words, students can use these blocks to obtain or change the value of a specific cell in the spreadsheets. The environment also provides blocks for automatically drawing charts

using a spreadsheet. One advantage of this programming environment is that various types of data are prepared in advance, so that students do not have to scrape the web themselves. However, when students create programs using the latest data or data related to their own interests, they must manually scrape the data themselves.

If there were "general" web scraping blocks that could be used for scraping a user-specified web page, it would help students to develop their own creative projects. In the following sections, we present novel web scraping blocks in order to address this problem.

3. Novel Scratch Programming Blocks for Web Scraping

In this section, we present novel programming blocks for web scraping. Our blocks were built on top of the "Tooee" block interfaces presented in our previous research [2]. In Section 3.1, we briefly introduce these block interfaces. In the following subsections (Section 3.2 through Section 3.6), we present five types of programming blocks for web scraping: (1) web scraping blocks, (2) web automation blocks, (3) mouse automation blocks, (4) keyboard automation blocks, and (5) file access blocks.

3.1. Block Interfaces for Communicating with WebSocket Servers

In our previous research [2], we defined novel Scratch block interfaces for communicating with WebSocket servers that can be implemented using text-based programming languages such as Python or JavaScript. The main block interfaces are demonstrated in Figure 2 as follows:

- The first block is the Scratch block that connects a Scratch client to a WebSocket server. Once connected, the connection continues until the program ends.
- The second and third blocks are the blocks that send messages to WebSocket servers. In the example shown in the figure, the second block sends the string "the weather" to the WebSocket server, and the third block sends the two strings "my score" and "10" to the WebSocket server.
- Whenever the first, second, and third blocks are executed, the Scratch client receives a response message from the WebSocket server, and the value is stored in the fourth block of this figure, i.e., "answer from Tooee".

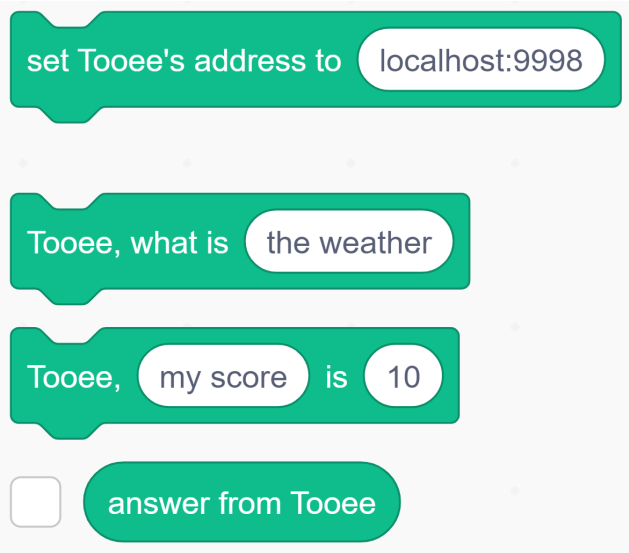

Figure 2. Scratch block interfaces (named Tooee [2]) for communicating with WebSocket servers. Our programming blocks were built on top of these block interfaces.

Note that these are just block interfaces, and how these blocks behave depends on how the WebSocket server is programmed. For example, when we execute the second

block in Figure 2, the Scratch client sends the string "the weather" to the WebSocket server. However, the WebSocket server decides what action the server performs when it receives this string. Some web servers may be programmed to send the weather in London when receiving this string, and some other web servers may be programmed to send the weather in Los Angeles.

Our main idea is that if we can implement a WebSocket server that performs web scraping tasks based on these block interfaces, then K-12 students should be able to easily create various types of web scraping applications using these blocks. We describe our approach in detail in the following subsections.

3.2. Web Scraping Blocks

One way to obtain specific content on a web page is to: (1) fetch the HTML file of the web page and (2) extract the content of an HTML element using the corresponding CSS selector. Obviously, if we use Python, we can perform these tasks very easily using popular libraries such as Selenium and Beautiful Soup. On the other hand, block-based programming languages such as Scratch cannot perform these tasks, because there are no corresponding programming blocks.

We implemented a novel WebSocket server in order to address this problem. First, we assume that Scratch users will send the URL of a web page with the reserved word "#open", as follows:

- Tooee, (#*open*) is (https://dblp.org/)

Whenever the WebSocket server receives two strings and the first string contains "#open", the server runs the Chrome web browser and opens the web page specified in the second string (in this case, https://dblp.org). Here, if the Scratch client and a Python server are different computers, the client can remotely control the server computer. In order to prevent unexpected accidents caused by remote control, by default we force the server to open the web browser only if both the client and server programs run in the same computer.

After opening the web page, a Scratch user can send another request to obtain the content of a specific HTML element. For example, if the HTML file has *<dt id="blog-532">RDF</dt>*, then the user can obtain the content "RDF" by sending the appropriate CSS selector with the reserved word "#scrape":

- Tooee, (#*scrape*) is (#*blog-532* > *a*)

Whenever the WebSocket server receives two strings and the first string is "#scrape", the server extracts the content of the HTML element using the CSS selector specified in the second string. After that, the server sends the extracted content to the Scratch client, and the client stores the content in the Tooee variable "answer from Tooee".

One of the most difficult tasks for students when using this block is to write the appropriate CSS selector of a HTML element. Fortunately, the students do not have to learn how to write the selector, because Chrome Developer Tools provides it automatically. All the students have to do is simply: (1) press *CTRL* + *SHIFT* + *C* to open the Developer Tools in Inspect Element mode, (2) click an HTML element, and (3) copy the CSS selector of the HTML element by right-clicking the selected HTML code. For example, "#blog-532 > a" can be copied to the clipboard through this process.

Figure 3 shows example Scratch code for scraping a DBLP web page. After running this code (clicking the Green Flag and pressing the space key), the "answer from Tooee" block will have the string "RDF". Note that in this code, the reserved words are encapsulated by the Scratch variables in order to make it easy for students to use the web scraping blocks. Here, the variable "URL to open" has "#open", and the variable "selector to scrape" has "#scrape".

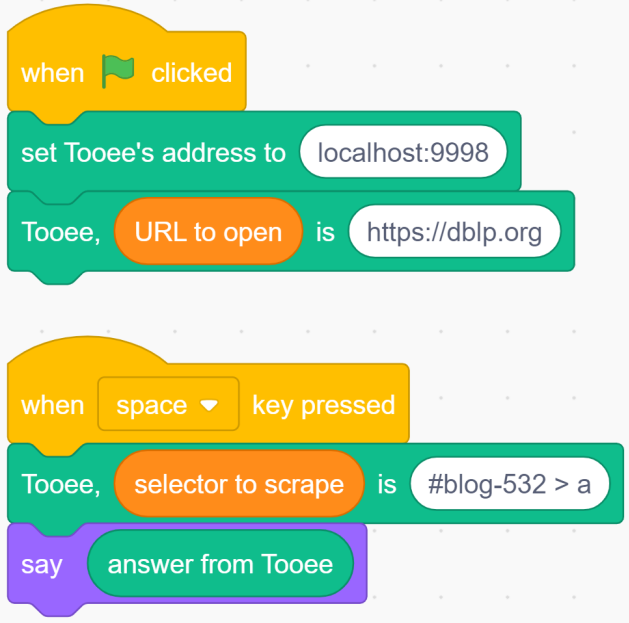

Figure 3. An example Scratch code for scraping a DBLP web page. When the green flag is clicked, the Scratch client connects to the WebSocket server and opens the https://dblp.org web page. When the space key is pressed, the content of the HTML element corresponding to the CSS selector "#blog-532 > a" is scraped.

3.3. Web Automation Blocks

It can also be helpful for students in scraping various types of websites if "web automation" blocks are available. For example, assume that students want to scrape news articles. Because many news websites assign a different URL for each news article, students need an automatic way of discovering these URLs by clicking suitable HTML elements. As another example, if a restaurant website provides a dynamic web page, and additional information appears only when students scroll down the web page, then they also need an automatic way to scroll down.

In this subsection, we present two web automation blocks. The first block is for automatically clicking an HTML element, and its block interface is as follows:

- Tooee, (*#click*) is (*XPath*)

This block interface is similar to the block interfaces introduced in Section 3.2, except that the reserved word is "#click" and XPath is used instead of a URL or a CSS selector. If a WebSocket server receives two strings and the first string is "#click", then the server clicks the HTML element specified in the second string.

The second block is for automatically scrolling down a web page, and its block interface is as follows:

- Tooee, (*#scroll*) is (*XPath*)

Here, only the reserved word is changed. Whenever a WebSocket server receives "#scroll" as the first string, it scrolls down the web page using the XPath expression described in the second string. Note that some dynamic web pages are so complex that students need to scroll down the web page multiple times in order to see a new HTML element.

Figure 4 illustrates an example application using these web automation blocks. Here, we assume that the web page "https://dblp.org" has already been opened on the browser by executing the two scripts shown in Figure 3 and that the variable "XPath to click" has "#click" and the variable "XPath to scroll" has "#scroll". When the "c" key is pressed, the WebSocket client sends the two strings "#click" and "//*[@id="news"]/div/p/a/em". When the WebSocket server receives "#click" as the first string, it clicks the HTML element

described by the second string. For example, if the HTML file has "<div id="news" ...><div ...> <a ...>more blog posts ", then the anchor text "more blog posts" will be clicked when the "c" key is pressed. Similarly, when the "down arrow" key is pressed, the WebSocket server receives the two strings "#scroll" and "/html". Because the first string is the reserved word "#scroll", the server scrolls down the HTML element described in the second string (in this case, "/html").

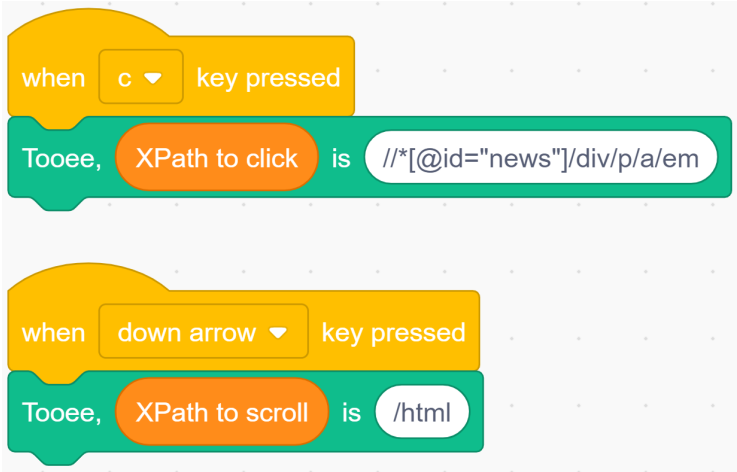

Figure 4. An example program that clicks or scrolls the specified html elements corresponding to the XPaths.

3.4. Mouse Automation Blocks

The proposed blocks described above are related to handling HTML elements. However, sometimes we need an automatic way to control the mouse using mouse cursor coordinates rather than HTML elements. For example, if we want to save an image shown on the HTML file and there is no button for saving the image, we must place the mouse cursor on the image and right-click on it.

Therefore, in this subsection, we additionally present novel mouse automation blocks that use mouse cursor coordinates rather than HTML elements. First, we propose the following block to discover the current mouse cursor position:

- Tooee, what is (*#pos*)

Here, we only send the reserved word "#pos". After executing this block, the current x and y coordinates of the mouse cursor are stored in the "answer from Tooee" block. For example, when the x and y coordinates of the mouse cursor are 1639 and 549, respectively, the "answer from Tooee" block will have "1639 549" after executing this block. Figure 5 shows example code using these blocks. Here, we assume that the variable "mouse x y" has "#pos".

If we know the x and y coordinates, then we can control the mouse by using them, as in the following blocks:

- Tooee, (*#click*) is (*x y*)
- Tooee, (*#right*) is (*x y*)
- Tooee, (*#double*) is (*x y*)

Here, we denote the x and y coordinates as x and y. When the WebSocket server receives the reserved word "#click", it clicks the left mouse button at the position corresponding to the given x, y coordinates. Similarly, the reserved word "#right" indicates right-clicking the mouse, and the word "#double" indicates double-clicking the mouse.

The block interface for dragging and dropping the mouse is as follows. Here, the reserved word "#drag" is used, and both the start position (x1 and y1) and the end position (x2 and y2) must be sent to the WebSocket server.

- Tooee, (*#drag*) is (*x1 y1 x2 y2*)

Figure 6 shows example code for mouse automation. Here, we assume that the variables "position to click", "position to right-click", "position to double-click", and "position to drag-and-drop" have "#click", "#right", "#double", and "#drag", respectively. Thus, when we execute this program, it clicks, right-clicks, and double-clicks the mouse on coordinates (1639, 549), and drags and drops the mouse from coordinates (270, 750) to (758, 924).

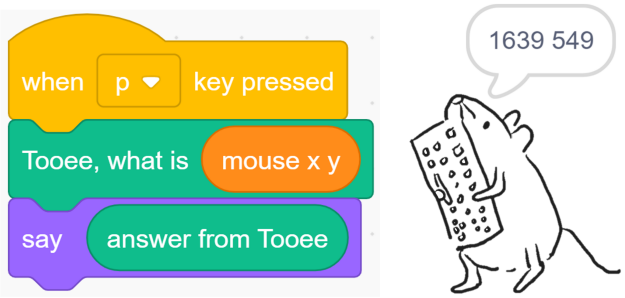

Figure 5. Example code to discover the current mouse cursor coordinates for mouse automation. In this example, the "answer from Tooee" block has a value of "1639 549".

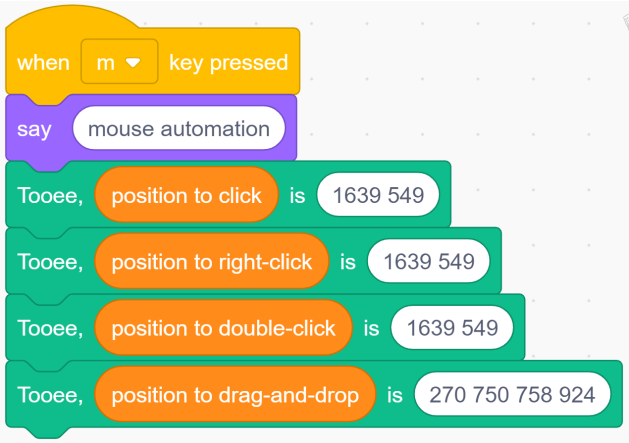

Figure 6. An example program that automates the mouse using the mouse cursor coordinates. In this example, the program performs a total of 4 operations: (1) "click", (2) "right-click", (3) "double-click", and (4) "drag-and-drop".

3.5. Keyboard Automation Blocks

We present two types of blocks for keyboard automation. The first is to enter a key (including a hot key) using the reserved word "#key", and the second is to enter a string using "#input". The block interfaces of these blocks are as follows:

- Tooee, (*#key*) is (*a key or a hot-key*)
- Tooee, (*#input*) is (*a string*)

Figure 7 shows example code for keyboard automation. Here, the variables "key to press" and "string to input" have "#key" and "#input", respectively. The program code is executed as follows. First, the two hot keys "CTRL + C" and "CTRL + V" are entered sequentially. That is to say, if students use a Windows operating system, the currently selected string is copied and pasted. Second, the string "programming" is entered.

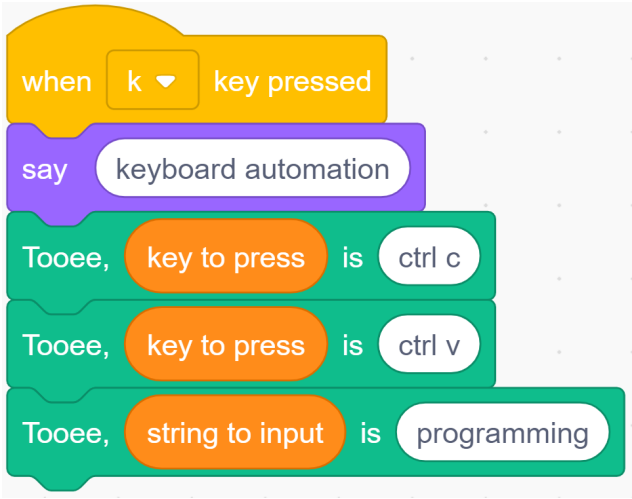

Figure 7. Example Scratch code to press the hot keys *CTRL* + *C* and *CTRL* + *V* and enter the character string *programming*.

3.6. File Access Blocks

Scratch does not provide file access blocks by default. However, it is often necessary to save the scraped data to a file and load it when needed. Thus, we present blocks that can write and read files. One challenge is to develop a way to make file access easier for K-12 students.

Our idea is to store students' data as key–value pairs. Here, keys are unique, and each key has a corresponding unique value. If a block does not have the aforementioned reserved words, it is regarded as a file access block. The following block interfaces represent saving data to a file and loading data from a file, respectively.

- Tooee, (*key*) is (*value*)
- Tooee, what is (*key*)

Figure 8 shows example code for saving and loading data. In this program, when the *s* key is pressed, a pair with "title" as the key and "programming" as the value are saved in the specific file. When *l* is pressed, the value corresponding to the key "title" is loaded from the file and stored in the "answer from Tooee" variable.

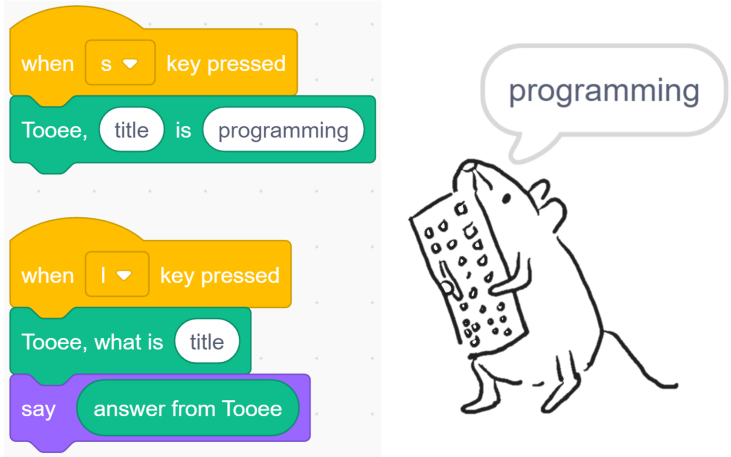

Figure 8. An example program for storing and retrieving data in the form of key–value pairs.

4. Experiments

4.1. Experimental Setup

We conducted two lectures for K-12 teachers. The curriculum for the two lectures was the same, with six teachers taking the first class and nine teachers taking the second class. The lectures were held online due to the coronavirus pandemic. A total of 15 online videos were uploaded, and each video was about 1 h and 15 min long. In these lectures, teachers communicated through our university's LMS. Teachers were free to ask questions, and based on what they had learned, they created their own projects as final assignments and shared them with each other through the LMS.

In these lectures, teachers were trained to implement the following example applications using the blocks introduced in Section 3. Note that the fifth, sixth, seventh, ninth, and tenth applications were inspired by the programs shown in the GitHub repository for data analysis projects [26]:

1. A "popdog" program using the file access blocks. When a user clicks on a puppy that appears on the screen, the score is incremented and saved to a file. When the program is restarted, the score saved in the file is loaded.
2. A coronavirus dashboard program using "#open" and "#scrape". When the program starts, it opens the web page on the specific portal site that shows the number of confirmed coronavirus cases. The program scrapes this number using the appropriate CSS selector and displays it on the screen.
3. A program that aggregates the weather forecasts from various weather forecast websites using "#open", "#click", and "#scrape".
4. A program that scrapes movie information provided by a movie website and ranks them based on movie ratings or cumulative audiences using "#open", "#click", and "#scrape".
5. A program that shows the overall ranking by scraping the rankings of several music sites using "#open", "#click", and "#scrape". Because the music sites are usually updated every hour, there is no need to scrape data every time the program is executed. Therefore, in order to speed up the program execution, it is implemented in such a way that the scraped data are saved into a file and loaded whenever necessary using our file access blocks.
6. A program that visualizes the statistics of an online video platform using "#open", "#click", and "#scrape".
7. A program that visualizes the number of coffee shops of a specific brand by region using "#open", "#click", "#scrape", and "#scroll". In the coffee shop website we scraped, we used both the "#click" and "#scroll" reserved words because we had to click on several menus sequentially and scroll down to see the information.
8. A program to automatically search and download cat images through a search engine using "#open", "#click", "#right", "#input", and "#key". Here, our mouse automation blocks are needed to bring up the save menu, and our keyboard automation blocks are also needed to enter a file name and press the enter key.
9. A program that visualizes the frequencies of hashtags on a specific community website using "#open", "#click", and "#scrape".
10. A program that scrapes product information for wireless vacuum cleaners from a specific shopping mall website using "#open" and "#scrape" and recommends products that meet its user's requirements.

We conducted an online anonymous survey of teachers after these lectures. Of the 15 teachers, 14 were primary school teachers and one was a secondary school teacher. Although the primary school teachers did not major in computer science, they were trained at their universities to teach computer science to primary school students. The survey consisted of three parts. The first part consisted of questions asking about personal programming and educational experience as follows:

- How long have you been studying block-based programming languages (such as Scratch or App Inventor)?
- How long have you been studying text-based programming languages (such as Python or JavaScript)?
- How long have you been teaching programming to students?
- If you have studied web scraping (or similar topics), briefly describe what you studied.

The second part consisted of the following questions asking how effective our blocks are from the perspective of K-12 students. The first three questions are related to Research Question 1 presented in Section 1, and the last two questions are related to Research Question 2:

- Question 1. Do you think K-12 students could implement these web scraping programs easily?
- Question 2. Do you think K-12 students will be interested in these web scraping programs?
- Question 3. Do you think these web scraping programs are practical programs?
- Question 4. Do you think it will be helpful for K-12 students to improve their programming skills by implementing these web scraping programs?
- Question 5. Do you think it will be helpful for K-12 students to improve their data literacy skills by implementing these web scraping programs?

In order to verify the effectiveness of our blocks, we compared our blocks with the Entry spreadsheet blocks. Thus, the third part consisted of questions asking how effective the Entry spreadsheet blocks were in terms of the five evaluation measures. Before asking these questions, we introduced the official example applications that exploit the Entry spreadsheet blocks. We did not consider Machine Learning for Kids or Cognimates in this survey, because these programming environments provide very few example applications related to web scraping.

Note that the questions were designed for responses on a five-point Likert scale (strongly disagree, disagree, neutral, agree, and strongly agree), and teachers freely expressed their opinions on the strengths and weaknesses of these blocks. The survey results are described in the next subsection.

4.2. Experimental Results

All teachers who attended the lectures responded to our anonymous survey. Figure 9 summarizes the teachers' programming and educational experiences. As shown in Figure 9, most of the teachers (13 out of 15) had experience in studying both block-based and text-based programming languages. Of the two remaining teachers, one teacher had experience with only text-based programming languages, and the other teacher had experience with only block-based programming languages. In addition, many teachers (11 out of 15) had experience in teaching programming to their students. All teachers responded that they had no web scraping experience before attending the lectures. However, some teachers responded that they had a great deal of experience with other programs that manipulate data, such as Entry, Scratch, Machine Learning for Kids, and Teachable Machine. One teacher had experience with data visualization using Tableau.

Table 1 shows the experimental results comparing the Entry spreadsheet blocks and our web scraping blocks in terms of ease of use (question 1), degree of interest (question 2), practicality (question 3), programming skills (question 4), and data literacy (question 5). Here, the numbers outside parentheses represent means, and numbers inside parentheses represent standard deviations. For example, our blocks had a mean score of 4.87 and a standard deviation of 0.35 for the third question.

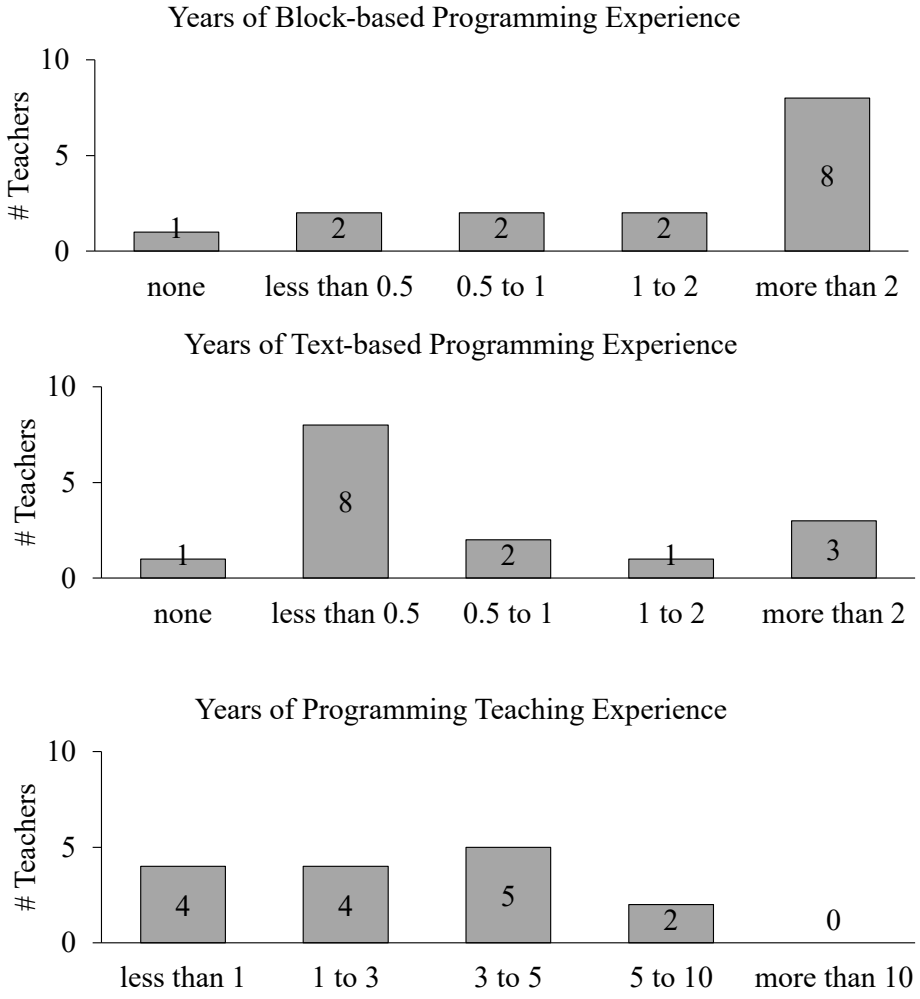

Figure 9. Summary of the teachers' programming and educational experience.

Table 1. Experimental results comparing the Entry spreadsheet blocks and our web scraping blocks for the five evaluation measures. Numbers outside parentheses represent means, and numbers inside parentheses represent standard deviations.

Evaluation Measure	Entry Spreadsheet Blocks	Our Web Scraping Blocks	*p*-Value
Ease of Use	3.47 (0.99)	**3.73 (0.96)**	0.24821
Degree of Interest	4.07 (0.96)	**4.27 (0.88)**	0.17971
Practicality	4.33 (1.05)	**4.87 (0.35)**	0.03389 **
Programming Skills	4.87 (0.35)	**4.93 (0.27)**	1.00000
Data Literacy	4.73 (0.59)	**4.86 (0.53)**	0.31731

** $p \leq 0.05$.

We performed a Wilcoxon signed-rank test with an alpha level of 0.05. The experimental results (Figure 10) show that our web scraping blocks did not show a statistically significant difference compared to the Entry spreadsheet blocks, except for the practicality evaluation measure. However, note that our blocks all showed higher average scores for all evaluation measures. In particular, our blocks showed very high scores (over 4.8 points) for the third, fourth, and fifth evaluation measures. A more detailed analysis of the experimental results is presented in the next section.

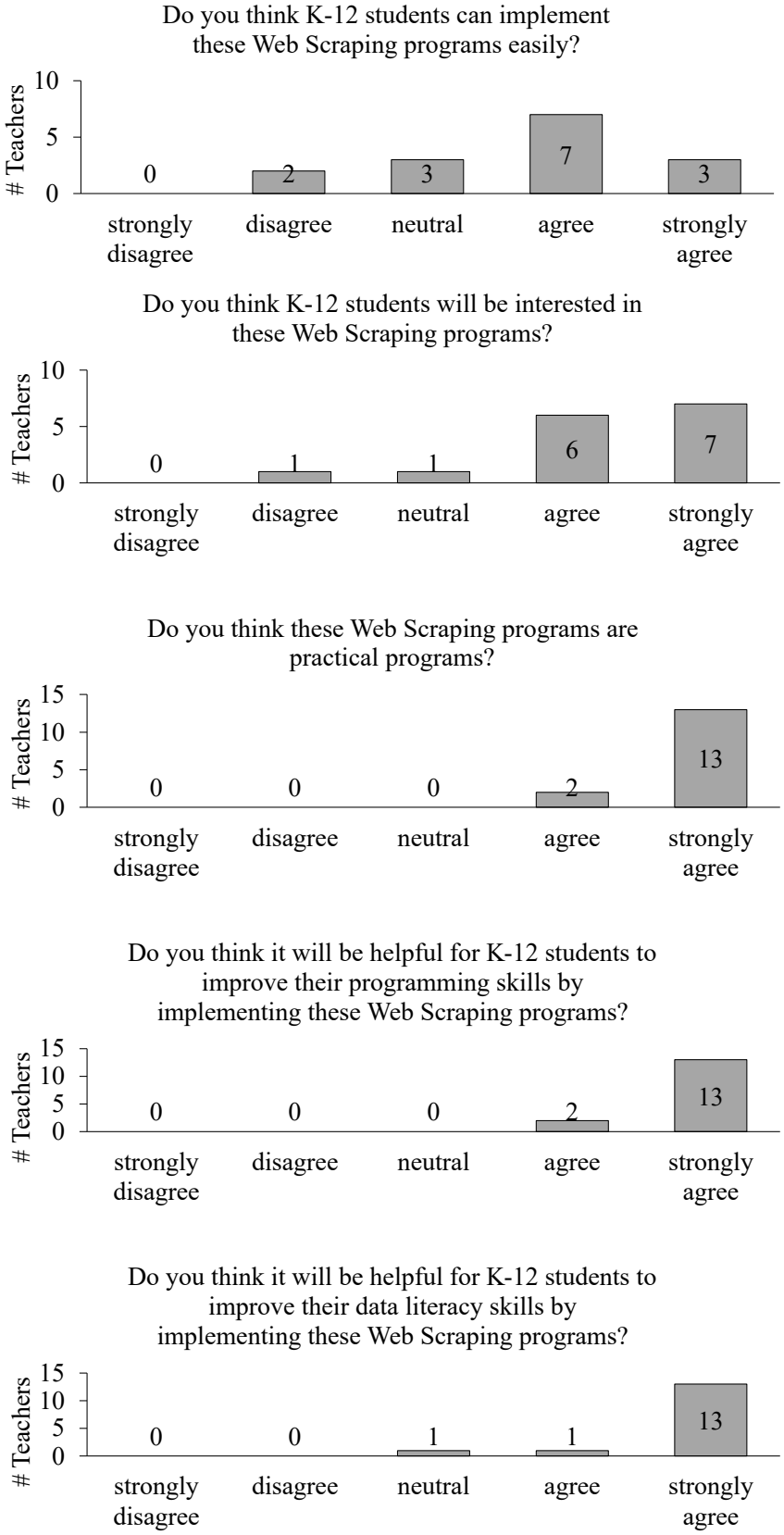

Figure 10. The teachers' evaluation results for the proposed programming blocks from the five perspectives.

4.3. Analysis of Results

In this subsection, we analyze the survey responses in detail. We describe in detail the evaluation scores given by the teachers, and also show their comments on our web scraping blocks.

- Question 1. Do you think K-12 students can implement these web scraping programs easily?

A total of 10 out of the 15 teachers stated that K-12 students would be able to easily create web scraping programs. More specifically, the teachers stated that it would be easier for K-12 students to perform web scraping with a block-based programming language instead of Python. They also stated that the proposed web scraping blocks were intuitive to use.

One teacher answered "neutral" to this question, expressing the opinion that although primary school students would find it difficult to create these applications, secondary school students could easily create them. On the other hand, one teacher who answered "strongly agree" presented the opposite opinion, stating that the proposed blocks were suitable for primary school students in grades four, five, and six.

The teachers who answered "agree" described their reasons for not answering "strongly agree" as follows. (1) Firstly, while the proposed blocks are easy to use, a high level of Scratch programming skill is required for students to create the example applications. (2) Secondly, while there are websites where web scraping can be performed easily, there are also websites where it is very difficult to perform web scraping. Therefore, it would be better if the teachers had a high level of programming skill, in order to help their students easily. (3) Lastly, the teachers described external obstacles such as computers in their schools that were too slow to use the Scratch programming language.

We give the additional comments of the teachers below:

> "After taking this class, I personally conducted a web-scraping-based programming class for primary school students in grades 1–3. At first, some students found web scraping quite difficult. I think it will be easy for primary school students in grades 4–6 to learn web scraping."

> "The web scraping blocks seem easier because they are simpler and more intuitive than the Entry spreadsheet blocks."

> "Because data are scraped in a block programming language rather than a difficult language such as Python, I think students will be able to program with interest."

> "The advantage of the web scraping blocks is that students are free to scrape data by using them. However, the difficulty of programming will vary greatly depending on which websites the students scrape. If the blocks are used in a classroom setting, a teacher's guidance is required."

> "It seems that students should have basic programming skills, and rather than just getting them to start programming right away, it seems necessary to explain to them what the web scraping process is and how they can use it."

- Question 2. Do you think K-12 students will be interested in these web scraping programs?

To this question, most teachers (13 out of 15) answered "agree" or "strongly agree". The teachers stated that they expected to elicit their students' voluntary participation, because the students can scrape the real-life data they want on their own. The teachers also described that the proposed blocks allowed students not only to scrape data but also to visualize it in a fun way.

We give the additional comments of the teachers below:

> "I think that this educational material has the advantage of easily collecting data and visualizing it in an interesting way."

"The web scraping blocks have the huge advantage of allowing students to scrape their own fields of interest. I think it's great that students can get the latest data they want."

"Web scraping blocks can generate explosive interest from students and have the advantage of automating data collection."

- Question 3. Do you think these web scraping programs are practical programs?

All teachers agreed with this question, and 13 out of 15 teachers strongly agreed. Most of the teachers answered that our example applications are practical because they are very relevant to "real life" and use "real data" and "real-time data". One teacher suggested that it would be easier to create practical programs if additional blocks to automate visualization were presented. Another teacher responded that if additional educational materials for teachers were provided, they would be better able to guide students in creating practical programs.

We give the additional comments of the teachers below:

"I think it's good that educational materials using web scraping blocks are closely related to our real life."

"The web-scraping-based programs are more practical than the Entry-spreadsheet-based programs because they are more relevant to our real life."

"The Entry spreadsheet blocks are suitable for learning the basic principles of data, and web scraping blocks have the advantage of being able to utilize real-life data and real-time data."

- Question 4. Do you think it will be helpful for K-12 students to improve their programming skills by implementing these web scraping programs?

When we asked the teachers if creating these web scraping programs would help students improve their programming skills, 13 out of 15 teachers responded that they would be very helpful. There were two teachers who answered "agree" instead of "strongly agree", one with no block-based programming experience and the other with less than six months of block-based programming experience. Through this result, it can be inferred that the proposed blocks are more effective for teachers (or students) with a great deal of experience in using block-based programming languages.

- Question 5. Do you think it will be helpful for K-12 students to improve their data literacy skills by implementing these web scraping programs?

Most of the teachers (13 out of 15) answered that they strongly agreed with this question. The teachers responded that the proposed blocks would be very effective for data literacy education in that students could scrape data in their field of interest on their own and interpret the scraped data from various perspectives. Two teachers responded "neutral" or "agree", and they had no, or less than six months of, experience with block-based programming languages. Therefore, it is expected that data literacy education will be more suitable for teachers (or students) with more than six months of block-based programming experience.

One teacher commented on this question as follows:

"It is essential to collect data for training artificial intelligence models. Therefore, I believe that web scraping is an area of data literacy that is essential for the future society in which students will live."

4.4. Limitations of the Study

One limitation of this paper is that, as a result of the experiments, the p-values exceeded 0.05 in four out of the five evaluation measures. Although our blocks achieved higher average scores for all five evaluation measures, it will be necessary to conduct additional experiments by increasing the number of participants. Another limitation is that teachers gave relatively low scores for the "ease of use" evaluation measure compared to the

other evaluation measures. Since the teachers predicted that children in early primary school would find our web scraping blocks difficult to use, it will be necessary in future work to develop easier web scraping blocks, so that even very young students can easily exploit them.

5. Conclusions

In this paper, we presented novel Scratch programming blocks for web scraping. The blocks include not only web scraping blocks that can open specific web pages and scrape the contents of specific HTML elements using CSS selectors but also web automation blocks that can click or scroll specific HTML elements using XPaths. We also presented mouse and keyboard automation blocks. The mouse automation blocks consist of a block that discovers the current mouse cursor coordinates and blocks that can click, right-click, double-click, and drag-and-drop using the given mouse cursor coordinates. The keyboard automation blocks include a block that allows a key or a hot key to be pressed and a block that allows a character string to be entered. We also presented intuitive file access blocks that can store and retrieve data in the form of key–value pairs, enabling the reuse of data obtained through web scraping.

We conducted web scraping education using the proposed blocks through two sets of lectures over three weeks for a total of 15 teachers. The teachers created a total of ten web scraping applications through the lectures. Most of the teachers expected these blocks to be very effective in teaching web scraping to K-12 students. Based on the evaluation results from the teachers' responses, we draw the following conclusions. (1) It is expected that students will be able to easily create practical programs by using our web scraping blocks (Research Question 1). The teachers gave very high scores for the "practicality" of our blocks compared to the existing blocks ($p = 0.03389$). In addition, they agreed that our blocks were difficult for children in early primary school to use but not for upper-grade primary school students or secondary school students. (2) In addition, these blocks are expected to help improve students' programming and data literacy skills (Research Question 2). Surprisingly, all teachers with more than six months of block-based programming experience "strongly agreed" that K-12 students could build their programming and data literacy skills through creating our web scraping applications.

As far as we know, this is the first study intending to perform web scraping using a block-based programming language and the first paper on web scraping education for K-12 students. Therefore, there is a shortage of relevant educational materials that teachers can use to guide their students. As future work, we plan to develop various types of educational materials using our proposed blocks and release them on the http://tooee.org website. In addition, since our research did not show statistically significant results for four of the evaluation measures due to the small number of participants, we also plan to re-validate the experimental results through additional surveys.

Author Contributions: conceptualization, Y.P. and Y.S.; data curation, Y.P.; investigation, Y.P. and Y.S.; methodology, Y.P. and Y.S.; supervision, Y.S.; validation, Y.P. and Y.S.; writing—original draft, Y.P. and Y.S.; writing—review and editing, Y.P. and Y.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by an Incheon National University Research Grant in 2020. This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2020R1I1A3068836).

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

References

1. Schedlbauer, J.; Raptis, G.; Ludwig, B. Medical informatics labor market analysis using web crawling, web scraping, and text mining. *Int. J. Med. Inform.* **2021**, *150*, 104453. [CrossRef] [PubMed]
2. Park, Y.; Shin, Y. Tooee: A Novel Scratch Extension for K-12 Big Data and Artificial Intelligence Education Using Text-Based programming blocks. *IEEE Access* **2021**, *9*, 149630–149646. [CrossRef]

3. Resnick, M.; Maloney, J.; Monroy-Hernández, A.; Rusk, N.; Eastmond, E.; Brennan, K.; Milner A.; Rosenbaum, E.; Silver, J.; Silverman, B; et al. Scratch: Programming for all. *Commun. ACM* **2009**, *52*, 60–67. [CrossRef]
4. Maloney, J.; Resnick, M.; Rusk, N.; Silverman, B.; Eastmond, E. The Scratch programming language and environment. *ACM Trans. Comput. Educ.* **2010**, *10*, 16. [CrossRef]
5. ACM. CSTA K-12 Computer Science Standards. Available online: https://portal.ct.gov/-/media/SDE/CTE/CSTA-K12-ComputerScience-Standards-Revised-2017.pdf (accessed on 7 July 2022).
6. Wolber, D. App inventor and real-world motivation. In Proceedings of the 42nd ACM Technical Symposium on Computer Science Education, Dallas, TX, USA, 9–12 March 2011; pp. 601–606.
7. Wolber, D.; Abelson, H.; Spertus, E.; Looney, L. *App Inventor*; O'Reilly Media: Newton, MA, USA, 2011.
8. The Official Scratch Website. Available online: https://scratch.mit.edu (accessed on 21 May 2022).
9. The Official App Inventor Website. Available online: https://appinventor.mit.edu/ (accessed on 21 May 2022).
10. Fronza, I.; Corral, L.; Pahl, C. Combining block-based programming and hardware prototyping to foster computational thinking. In Proceedings of the 20th Annual SIG Conference on Information Technology Education, Tacoma, WA, USA, 3–5 October 2019; pp. 55–60.
11. Zhang, L.; Nouri, J. A systematic review of learning computational thinking through Scratch in K-9. *Comput. Educ.* **2019**, *141*, 103607. [CrossRef]
12. Dodero, J.M.; Mota, J.M.; Ruiz-Rube, I. Bringing computational thinking to teachers' training: A workshop review. In Proceedings of the 5th International Conference on Technological Ecosystems for Enhancing Multiculturality, Cádiz, Spain, 18–20 October 2017; pp. 1–6
13. Zhang, L.; Nouri, J.; Rolandsson, L. Progression of Computational Thinking skills in Swedish compulsory schools with block-based programming. In Proceedings of the Twenty-Second Australasian Computing Education Conference, Melbourne, VIC, Australia, 4–6 February 2020; pp. 66–75.
14. Grover, S.; Basu, S.; Bienkowski, M.; Eagle, M.; Diana, N.; Stamper, J. A framework for using hypothesis-driven approaches to support data-driven learning analytics in measuring computational thinking in block-based programming environments. *ACM Trans. Comput. Educ.* **2017**, *17*, 1–25. [CrossRef]
15. Arslan Namli, N.; Aybek, B. An Investigation of the Effect of Block-Based Programming and Unplugged Coding Activities on Fifth Graders' Computational Thinking Skills, Self-Efficacy and Academic Performance. *Contemp. Educ. Technol.* **2022**, *14*, ep341. [CrossRef]
16. Gleasman, C.; Kim, C. Pre-service teacher's use of block-based programming and computational thinking to teach elementary mathematics. *Digit. Exp. Math. Educ.* **2020**, *6*, 52–90. [CrossRef]
17. Grover, S.; Pea, R. Computational thinking in K-12: A review of the state of the field. *Educ. Res.* **2013**, *42*, 38–43. [CrossRef]
18. Yadav, A.; Mayfield, C.; Zhou, N.; Hambrusch, S.; Korb, J.T. Computational thinking in elementary and secondary teacher education. *ACM Trans. Comput. Educ.* **2014**, *14*, 5. [CrossRef]
19. Barr, D.; Harrison, J.; Conery, L. Computational thinking: A digital age skill for everyone. *Learn. Lead. Technol.* **2011**, *38*, 20–23.
20. Wing, J.M. Computational thinking. *Commun. ACM* **2006**, *49*, 33–35. [CrossRef]
21. Wing, J.M. Research notebook: Computational thinking—What and why. *Link Mag.* **2011**, *6*, 20–23.
22. Wing, J.M. Computational thinking's influence on research and education for all. *Ital. J. Educ. Technol.* **2017**, *25*, 7–14.
23. Druga, S. Growing up with AI: Cognimates: From coding to teaching machines. Ph.D. Thesis, Program in Media Arts and Sciences, Massachusetts Institute of Technology, Cambridge, MA, USA, 2018.
24. Lane, D. *Machine Learning for Kids: An Interactive Introduction to Artificial Intelligence*; No Starch Press: San Francisco, CA, USA, 2021.
25. The Entry Programming Environment. Available online: https://playentry.org (accessed on 7 July 2022).
26. A GitHub Repository for Data Analysis Projects. Available online: https://github.com/Play-with-data/datasalon (accessed on 21 May 2022).