Evaluation of Block-Based and Text-Based Coding

BERNÁT Péter

Abstract. Beginner programmers usually have difficulties in creating syntactically correct programs. Some cases it is facilitated by features that help to enter text code, and in other cases by block-based programming that replaces conventional text-based coding. In my article, by comparing and evaluating the two possible types of coding, I prove that it is easier for beginners to learn block-based programming, but advanced programming requires transition to text-based coding. This will also be confirmed by three interviews with IT teachers. Then, I will point out that transition is possible within the same programming area and I will refer to my interviewees' experiences with the transition.

Keywords: teaching programming, block-based coding, text-based coding, interview

1. Block-based and text-based coding

The creators of educational programming languages aim at making the very first steps of programming comprehensible and motivating as much as possible. According to a taxonomy [1] created by the instructors of Carnegie Mellon University these intentions can be associated with expressing programs, structuring programs, or understanding program execution.

Beginner programmers usually have difficulties in creating syntactically correct programs. Some cases it is facilitated by features that help to enter text code, and in other cases by block-based programming that replaces conventional text-based coding. In block-based programming the code can be constructed by snapping commands as visual blocks together. These blocks can have either icons or text on them (Figure 1).

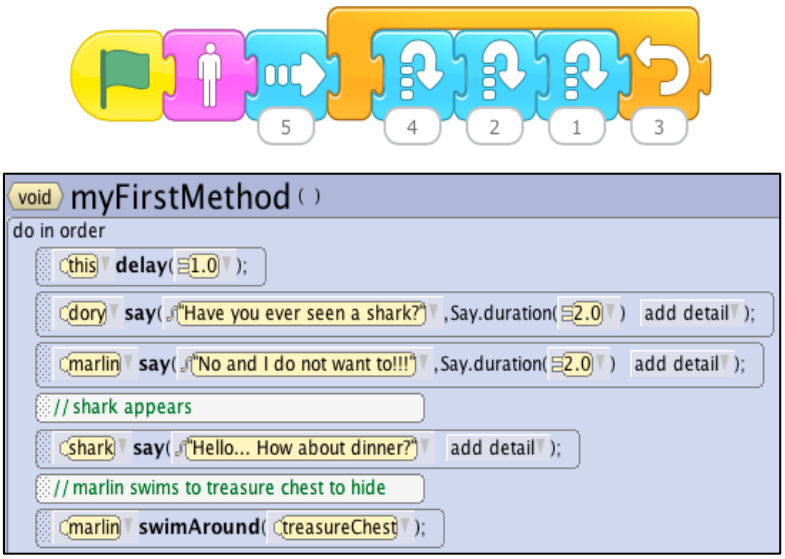


Figure 1: A code in a block-based programming language that uses icons (ScratchJr) and text (Alice)

In my article, by comparing and evaluating the two possible types of coding, I prove that it is easier for beginners to learn block-based programming, but advanced programming requires transition to text-based coding. This will also be confirmed by three interviews with IT teachers. Then, I will point out that transition is possible within the same programming area and I will refer to my interviewees' experiences with the transition.

When comparing, I always list in parentheses those educational programming environments that provide the feature in question. I examined ScratchJr (scratchjr.org), Scratch (scratch.mit.edu) and Alice (alice.org) block-based, and Imagine Logo (logo.sulinet.hu), RoboMind (robomind.net) and Small Basic (smallbasic.com) text-based programming languages.

2. Comparison of block-based and text-based coding

2.1. Inserting a language element into the code

In block-based programming environments, language elements that can be inserted into the programming area are available in categories of blocks grouped by function (ScratchJr, Scratch and Alice) (Figure 2). This eliminates the need to remember their names accurately and they can be added to the program without typing errors.

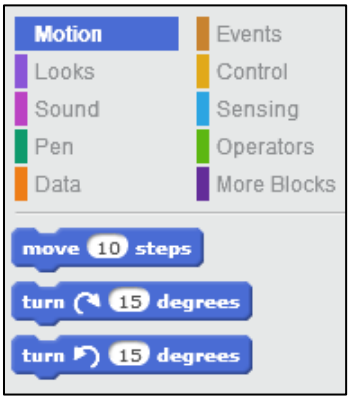


Figure 2: Categories of Scratch blocks, and some commands of the Motion group

In some environments, entering text code is aided by automatic code completion (Small Basic). This feature, based on the context and the characters already typed, offers a drop-down list of possible keywords and identifiers that can be easily inserted (Figure 3). It also helps to recall language elements, but misspellings can occur, and not all programming environments offer this feature.

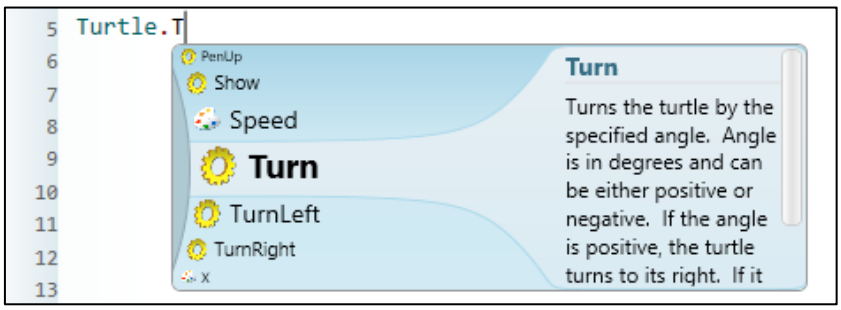


Figure 3: Automatic code completion in Small Basic

2.2. Giving arguments to a language element

In block-based languages, an element with its arguments form a single block, so only the right *number* of arguments can be given at the appropriate *positions* (prefix or infix) (ScratchJr, Scratch and Alice), and their *role* is also clear from the content of the element (Scratch and Alice) (Table 1, first column).

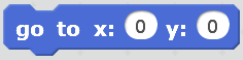
Scratch	Imagine Logo
	`stamp`
	`setpencolour`
	`setxy`
	`word`
	`and`

Table 1: Language elements that perform the same role in Scratch and Imagine Logo; in the former language the number, location and role of the arguments turn out, but not in the latter one

Also, one of the following options ensures that only a value from the right *type* can be given as an argument:

- the shape of the argument area specifies the type (for example, in the first column of the table, only two logical values can be inserted into the *and* operation) (Scratch);
- only values from the appropriate type can be typed in (for example, in the first column of the table, only numbers can be typed into the *go to* block) (ScratchJr, Scratch and Alice);
- the possible values can be chosen from a drop-down list (Scratch and Alice).

In text-based programming languages, elements do not express whether they need an argument and, if so, *how many,* from what *type,* in which *position* and for what *role* (Table 1, second column). Therefore, in some environments, a form can be opened in which the language element at the cursor can be parameterized, so that, just like in block-based coding, only the appropriate number and type of arguments can be given, whose role reveals from the form, and that are copied into the proper place by the environment (Imagine Logo) (Figure 4, left side).

Another part of text-based environments provides information about parameterizing the language element at the cursor in a context-sensitive help (Small Basic), but still too many or too few arguments may be given by mistake, or may be typed incorrectly (Figure 4, right side). And there are environments that do not offer either option (RoboMind).

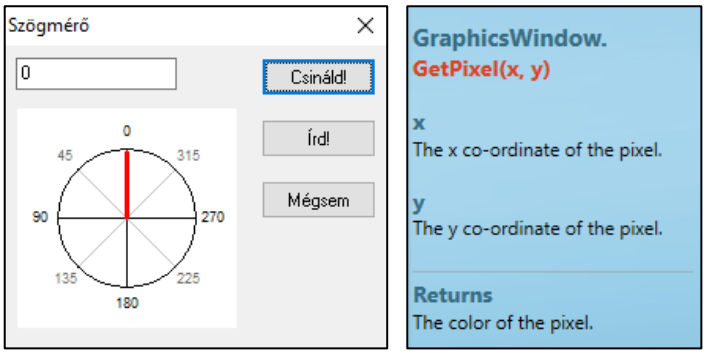


Figure 4: Form to help parameterize a command in Imagine Logo,
and context-sensitive help about parameterizing the command at the cursor in Small Basic

2.3. Creating a control structure

As for creating control structures, the situation is quite similar. In block-based languages, a control structure and its instruction blocks form a single block together, so only the appropriate number of instruction blocks can be given to it, which can only include instructions (ScratchJr, Scratch and Alice) (Figure 5).

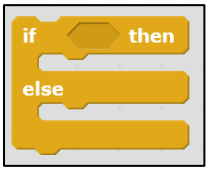


Figure 5: Selection in Scratch

While text-based coding, a context-sensitive help can inform the programmer of the number of required instruction blocks (Small Basic), but still too few or too many instruction blocks can be declared by mistake, in which the programmer can type not only instructions.

2.4 Syntactic correctness and indication of syntax errors

As mentioned above, only existing and well-parameterized language elements can be used in block-based programming, control structures can only be composed of a sufficient number of instruction blocks containing instructions only, and no delimiters are needed to separate instructions and arguments. That is why only syntactically correct programs can be composed of blocks. The programmer may face the fact that a step would cause a syntax error (for example, using a number as a logical condition) without making the error and getting an error message (ScratchJr, Scratch and Alice). Hence, making a mistake is not a failure, but the feedback is immediate (Figure 6).

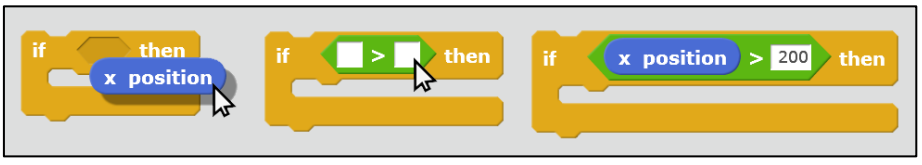


Figure 6: No numeric value, but a logical expression may be the condition of a selection in Scratch

Despite the help described earlier (if they are available at all), syntax errors can still occur in text-based programming. Errors are only indicated by error messages while starting or running the program (Imagine Logo, RoboMind and Small Basic). As feedback is delayed, the location and cause of the error are not always clear, and a list of error messages can be frustrating (Figure 7).

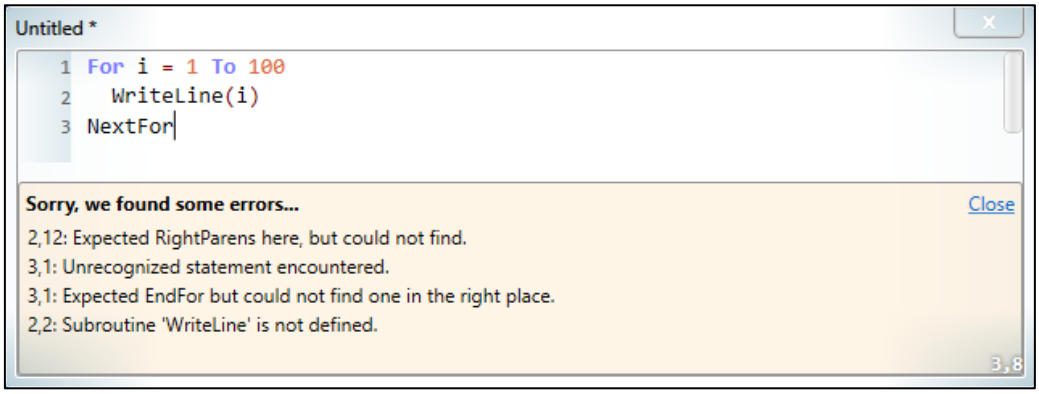
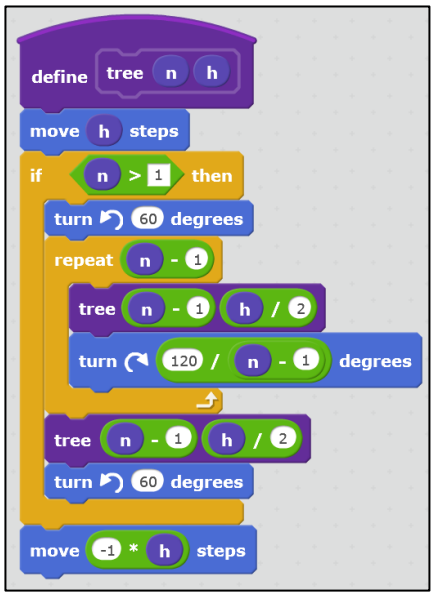


Figure 7: Error messages appear when starting the program in Small Basic

2.5 Readability

In block-based programming languages, even in complex expressions, it is clearly visible which instruction contain which values or sub-expressions as arguments (Scratch and Alice) (Figure 8, left). In text-based coding, parentheses usually help to read compound expressions (RoboMind and Small Basic), but you need to find the pairs of parentheses. In addition, there are some educational languages where the arguments are not enclosed in parentheses (Imagine Logo) (Figure 8, right).




```
to tree :n :h
forward :h
if :n > 1 [
left 60
repeat :n - 1 [
tree :n - 1 :h / 2
right 120 / (:n - 1)]
tree :n - 1 :h / 2
left 60]
back :h
end
```

Figure 8: Matching lines of a program that performs the same task in Scratch and Imagine Logo

In block-based languages, nesting of instruction blocks is also clear (ScratchJr, Scratch and Alice) (Figure 8, left). In text-based coding, it is commonly facilitated with different indentations. However, if the programmer does not indent the lines, the structure of instruction blocks becomes

more difficult to understand (Figure 8, right). Therefore, some programming environments offer the ability to automatically indent the lines (Small Basic and RoboMind).

2.6 Speed of entering code

In block-based environments, to insert a new language element, you must first select the appropriate category in the block set and then the language element in question, which must finally be dragged into the appropriate location in the program code (ScratchJr, Scratch and Alice). This is the case for each keyword, identifier, and even operator (Figure 9).

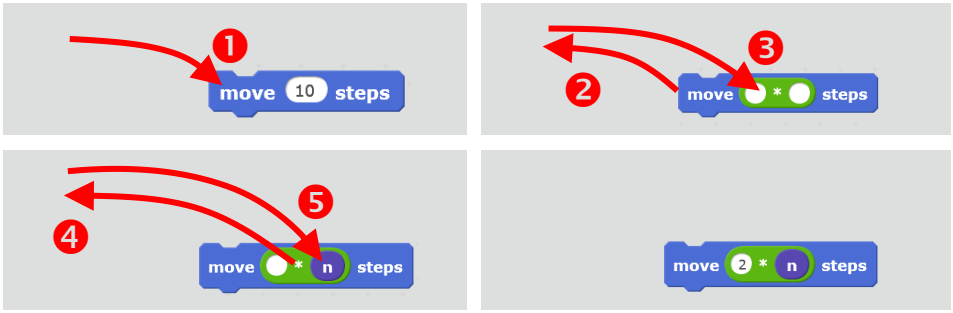


Figure 9: Mouse movements between the block set on the left side of the screen (not shown) and the programming area required to assemble the instruction shown in the lower right phase in Scratch

The same language elements can be typed faster in a text-based programming environment (Imagine Logo, RoboMind and Small Basic), even if the auto-completion feature is not available.

2.7 Extent of code

A program code made of blocks takes up significantly more space than its equivalent text code. In addition to the size of the blocks, this is also because the blocks can only be connected under each other (ScratchJr, Scratch and Alice), while in most text-based languages the instructions can be written on one line (Imagine Logo and RoboMind).

2.8 Editing in other environments

A block-based code can only be edited within the development environment and used only as an image outside (ScratchJr, Scratch, and Alice). In contrast, text codes can be modified in any code or text editor and then copied back to the development environment (Imagine Logo, RoboMind and Small Basic).

3. Evaluation of block-based and text-based coding

For novice programmers, block-based programming is more appropriate: primarily because it is not possible to make a syntax error, and the shapes of the blocks increase the readability of the program code. An additional advantage is that blocks can contain pictograms, making coding accessible to the youngest children, or longer expressions that are closer to everyday speech. Self-experimentation is also supported by the fact that all language elements are available in the block set. Research has also shown that block-based programming is easier for novice programmers [2],

and those who have previously studied block-based programming may achieve measurably better results in text-based programming [3].

On the other hand, it is not by accident that text-based coding is characteristic of professional programming: programmers experienced in using keyboard enter text faster by typing. For advanced programmers who have less trouble following syntactic rules and who produce a significant number of programs, text-based coding is more efficient.

4. Interviewees' opinions on block-based and text-based coding

An interview is a qualitative research method that allows for in-depth questioning and thus reveals perspectives, motivations, and correspondences for the researcher [4]. My first interviewee has been teaching 2–8 grade students for four years in a programming school in an educational environment in which both text-based and block-based coding is possible. The second is from an elite high school and teaches text-based programming in normal classes, but also block-based programming in her programming study groups. The third is also a computer science teacher at a reputable high school, however, only teaches text-based programming languages.

Without giving any specific points of view, I asked them to compare the two types of coding based on their experiences. The interviews were conducted anonymously, and for their processing I used the method of categorization [5]. According to this method, I grouped the arguments for block-based coding first and then for text-based coding.

4.1 Arguments for block-based coding

My interviewees stated that *language elements were more easily accessible* in block-based programming. "It is not necessary to know what instructions exist, they can be found in the menu, even without the help of a teacher." (Interviewee 3)

It was highlighted that it was easier to generate *syntactically correct code* from blocks. "Block programming is good because syntax errors cannot be made, only semantics; from this point of view it is better than text coding." (Interviewee 1) "It's easier to generate error-free code using blocks as a novice programmer." (Interviewee 2)

They spoke about the *better readability of the code* made of blocks. "In my opinion, block-based programming is clearer than text-based: for example, it is easier to see the relationship of nested loops or selections. The alignment of the code is flawless and regular, while, for example, there are several habits for aligning the text in text-based coding, some of which often make it difficult for me to read the code." (Interviewee 1)

It was considered important that blocks could be used for *replacing typing*. "It's a great relief for a lower grade student if she/he doesn't have to type instructions, only their parameters." (Interviewee 1) "Block-based languages are able to make programming available to young children who are not yet familiar with letters and numbers, and to young people who are not yet able to type efficiently, around 4–6 grade." (Interviewee 2)

4.2 Arguments for text-based coding

My interviewees also pointed out the advantages of text-based coding, such as *faster input*. "A very simple problem can be solved easily with blocks. However, as the complexity of the task increases, the effectiveness of this method reduces, as it becomes time-consuming to tinker up the solution with the mouse instead of typing." (Interviewee 1) "To write down a letter 'f' and then a number is much faster than searching for the *forward* instruction on a panel, dragging it to the screen with the mouse, and then typing the argument in the right place." (Interviewee 3)

They also highlighted that the *smaller extent of text codes* was more practical. "It's also very difficult to put together the code of a larger application in a block-based environment." (Interviewee 2) "If you already have a lot of elements on the screen, you can't get a good overview of the program in either remote or close-up view, and it's often not clear when and what happens. Of course, the text code can be long and does not necessarily fit on the screen, but at the same time it is more motivated to split the code into procedures that are enough to refer to in the right place. So, I think when a student is already above a certain level, text programming is more adequate." (Interviewee 3)

The experiences, arguments and counterarguments of my interviewees support my previous evaluation.

5. Transition from block-based to text-based coding

The easiest way of transition from block-based to text-based programming is using a development environment that supports conversion between the two types of coding. In such an environment the same (correct) code can be viewed and edited in both modes. An example is the micro:bit (microbit.org) minicomputer's Microsoft MakeCode (makecode.microbit.org), which allows users to switch between a block-based language and a text-based one built on the JavaScript syntax (Figure 10).

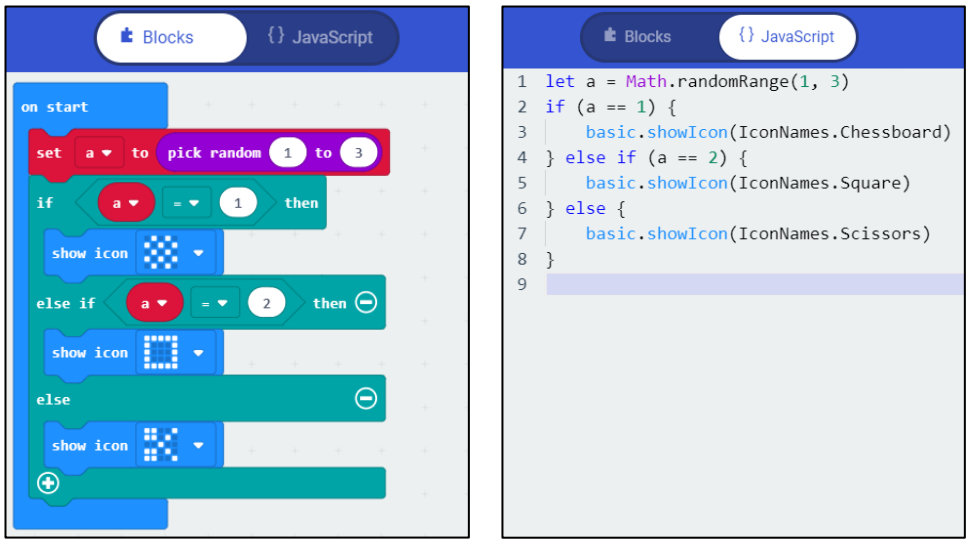


Figure 10: The same program in block and text view in Microsoft MakeCode

Of course, syntactically incorrect text code cannot be executed or converted back to block-based. In this case, programmer can request to restore the latest flawless version.

Transition from block-based to text-based programming is also possible even if the block-based environment does not allow to switch between the two types of coding. In order to have only one novelty at a time – the changed programming language – it is advisable to continue programming within the same area. For example, the Scratch and Imagine Logo programming languages provide this, since the former is block-based and the latter is text-based and in both it is possible to solve many types of exercises of turtle graphics (Figure 11). Both have objects (sprites and turtles) that have a pen and can be controlled by matching commands.

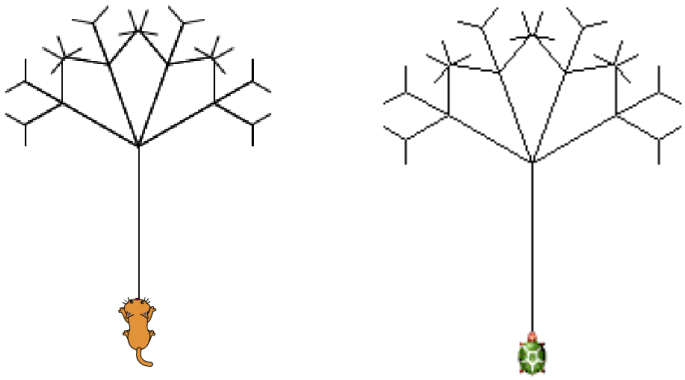


Figure 11: The result of the two matching programs shown in Figure 8 in Scratch and Imagine Logo

My publication *Solving Logo Contest Problems in the Scratch Programming Language* [6] can help the transition between the two languages, in which I compared the capabilities of the two programming languages in turtle graphics, and discussed the solution of a number of Logo Contest problems in Scratch, grouped by problem types.

6. Interviewees' experiences on transition

My first interviewee reported that students at their programming school "found text coding and its keywords very exciting and wanted to learn all very quickly" and "didn't miss block-based coding". This is probably because most students who attend that school are very receptive to programming.

However, the two high school teachers said that they often had to motivate their block-programmer students. "If you like block-based programming and you are keen on it, you don't really want to switch to text coding on your own. […] Some people, for example, realize that an algorithm to find prime numbers cannot be made effectively in Scratch. And then we can search together for a programming language that can be used more efficiently. However, in my experience, most Scratch users do not come to this idea by themselves." (Interviewee 2) "It was painful for him to switch, because he used to use blocks. And he didn't like typing very much." (Interviewee 3) My second interviewee, on the other hand, emphasized that a student who had previously programmed in a block-based environment "had a very good approach to programming and program design which was very useful in starting text-based programming. He had understood the control structures easily and accustomed to syntax rules without much difficulty."

7. Conclusion

In my article, I compared and evaluated the two types of coding independently of other features of educational programming languages, and I found that for beginners the block-based, while for advanced programmers, the text-based programming is more advantageous.

I believe that for beginners, regardless of age, block-based programming should be proposed, which is used in teaching introductory programming not only in public but also in higher education [7]. I personally believe that although text-based coding is clearly more efficient in advanced programming, the block-based view conveys the structure of the programming language better than plain text.

Bibliography

1. Kelleher, C. & Pausch, R. *Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers.* ACM Computing Surveys, 37(2), pp. 83-137, (2005) DOI: 10.1145/1089733.1089734
2. Weintrop, D. & Wilensky, U. *To block or not to block, that is the question: students' perceptions of blocks-based programming.* 14th International Conference on Interaction Design and Children, pp. 199-208, (2015) DOI: 10.1145/2771839.2771860
3. Armoni, M., Meerbaum-Salant, O. & Ben-Ari, M. *From Scratch to "Real" Programming.* Transactions on Computing Education, 14(4), (2015). DOI: 10.1145/2677087
4. Falus, I. & Ollé, J. *Az empirikus kutatások gyakorlata – Adatfeldolgozás és statisztikai elemzés.* (in Hungarian) Oktatás-kutató és Fejlesztő Intézet, Budapest, (2008).
5. Schleicher, N. *Kvalitatív kutatási módszerek a társadalomtudományban.* (in Hungarian) Századvég, Budapest, (2007).
6. Bernát, P. *Logo versenyfeladatok megoldása a Scratch programozási nyelven.* (in Hungarian) ELTE Informatikai Kar, Budapest, (2017). http://logo.inf.elte.hu/tanaroknak/logo-scratch%20v11.pdf
7. code.org: *Why top universities teach drag and drop programming*, (2014) https://www.youtube.com/watch?v=_Mwc1gc77dc

Authors

BERNÁT Péter

Eötvös Loránd University, Faculty of Informatics, Budapest, Hungary,
e-mail: bernatp@inf.elte.hu

About this document

Published in:

CENTRAL-EUROPEAN JOURNAL OF NEW TECHNOLOGIES IN RESEARCH, EDUCATION AND PRACTICE

Volume 2, Number 2. 2020

ISSN: 2676-9425 (online)

DOI:

10.36427/CEJNTREP.2.2.471

License