Computer Science and Information Systems 23(1):33–59 https://doi.org/10.2298/CSIS250401002K

Comparing Novice Programmers Performance with Block-based, Text-based, and Both Notations: A Study from Two Countries*

Tomaž Kosar[1], Srđa Bjeladinović[2], Dragana Ostojić[1], Milica S. Škembarević[2], Žiga Leber[1], Olga A. Jejić[2], Matej Moravec[1], Filip Furtula[2], Miloš D. Ljubisavljević[2], Ivan S. Luković[2], Marjan Mernik[1] and Matej Črepinšek[1]

[1] University of Maribor, Faculty of Electrical Engineering and Computer Science
Koroška cesta 46, Maribor, 2000 Maribor, Slovenia
{tomaz.kosar, dragana.ostojic, ziga.leber, matej.moravec, marjan.mernik, matej.crepinsek}@um.si
[2] University of Belgrade, Faculty of Organizational Sciences
Jove Ilića 154, Beograd, 11000, Belgrade, Serbia
{srdja.bjeladinovic, milica.skembarevic, olga.jejic, filip.furtula, milos.ljubisavljevic, ivan.lukovic}@fon.bg.ac.rs

Abstract. Teaching programming presents numerous challenges, one of which is selecting the most effective notation to introduce programming concepts to beginners. This study explores different notation approaches for learning fundamental programming concepts, with the objective of assessing how notation choice influences beginners' performance. To investigate this, we conducted a controlled experiment during short-term visits aimed at promoting programming in primary schools. Our multinational study divided participants into three groups: one using block-based notation (Poligot), one using text-based notation (Python), and one using both simultaneously (Poliglot). After completing a training session, the participants engaged in practical programming tasks to assess their performance in Python or the multiple-representation environment Poliglot. The results indicate that the choice of notation did not impact the performance of the participants significantly, as no statistically significant differences were found between the three groups. These findings were consistent across two different countries. Our study suggests that educators can use different notations (Python, Poliglot, or a combination) confidently when introducing programming to beginners. However, the performance results may improve when training sessions are extended over a longer period.

Keywords: Programming Education, Block-based Programming, Text-based Programming, Multiple-representation Environments, Programming Engagement, Novice Programmers.

1. Introduction

Introducing programming to children in primary school is crucial in today's digital world. Early exposure to programming not only equips children with valuable technical skills, but also enhances their complex problem-solving abilities [1, 2], creativity, communication, and teamwork skills. Despite its importance, informatics or programming subjects

* An extended version of The 19th Conference on Computer Science and Intelligence Systems/FedCSIS 2024 paper

are still not part of the primary school curriculum in many countries [3, 4]. This gap in primary school education can lead to an essential void in future job markets. By integrating programming into the curriculum [5], we can ensure that all children have the opportunity to develop these essential skills, preparing them for a future where technology, as we all agree, will play an important role in different aspects.

To address this deficit in primary school education we occasionally visit schools, spending time with the children demonstrating software development. During these visits, our aim is to engage the children by showcasing the development of simple applications and potential career opportunities in the field of Computer Science. We believe that, as guest lecturers in primary schools, we have several significant impacts. Guest lecturers can bring enthusiasm to the subject, which is different from the usual ones, making it more engaging for the children. The additional importance of guest lecturers is that they can pave the way for a creative approach to the adoption of basic programming by teachers in primary schools. Moreover, guest lecturers can introduce diverse perspectives and knowledge that might not be readily available through the standard curriculum, enriching the children's learning experience and highlighting the importance and relevance of programming skills in today's digital world.

However, teaching programming to beginners is a complex task that requires careful consideration of various factors. One of the key challenges is the limited time available to teach children programming during visits [6]. Another critical decision involves selecting the appropriate notation to introduce programming concepts [7, 8]. Using block-based notation (e.g., Scratch [9, 10], App Inventor [11]) allows the creation of visually appealing games and applications [12, 13]. Conversely, text-based notation enables focused teaching of fundamental programming concepts [14]. Each notation offers distinct advantages. We believe that block-based notation is the most inspiring way to introduce programming concepts, particularly for beginners. However, text-based notation is better suited for long-term learning, as it prepares students for professional programming, which relies primarily on textual coding. Early exposure to text-based syntax helps beginners build confidence and familiarity with industry-standard programming languages.

On the other hand, the transition from block-based programming to text-based programming is often highlighted as problematic; starting with block-based notation can lead novice programmers to be reluctant to switch to text-based notation [15]. This is a common pitfall that programming educators face. To address this problem, educational tools have emerged that enable educators to teach novice programmers to use both notations simultaneously [16, 17]. The most significant advantage of a multi-representational environment [18], which allows programming in two distinct notations, is that the transition from block-based to text-based notation occurs very naturally. For example, consider how time-consuming it is to write math equations using blocks; in contrast, text-based notation allows expressing math equations more naturally and efficiently. The strategy of presenting both notations simultaneously helps students integrate seamlessly and understand both forms of programming, easing the learning curve, and enhancing their overall comprehension. Using both notations simultaneously facilitates a smooth transition between the visual and textual programming, allowing students to leverage the strengths of each approach. This is especially beneficial for those who are already familiar with one notation but need to transition to the other. Although we developed one such multi-representational environment called Poliglot [18] that enables programming with both

notations simultaneously, we are still determining the advantages of using such an educational programming environment.

Regardless of the strengths of using each notation mentioned above, our motivation for this study arose from a practical need to evaluate whether the choice of programming notation has a meaningful impact on beginners' performance during short, introductory programming sessions. Specifically, our objective was to investigate whether using block-based, text-based, or both simultaneously (block and text) leads to statistically significant differences in participants' performance on basic programming tasks. To ensure a fair comparison, all the groups received the same amount of instructional and training time, although the participants using both notations were introduced to two notations simultaneously, potentially sacrificing performance. Furthermore, to examine how different educational contexts and prior programming exposure could influence the results, we conducted the study in two different countries as part of a broader multinational investigation. The following research questions guided our empirical study:

- RQ1: Does the type of programming notation (block-based, text-based, or using both simultaneously) influence programming performance significantly?
- RQ2: How does programming performance vary between two different national contexts?
- RQ3: How does the duration of programming training sessions affect performance outcomes, regardless of the notation used?

To test these research questions, we designed a controlled experiment [19–21], conducted in classrooms during short-term visits (2 hours), including training and a brief test at the end of the sessions. Additionally, in one country, we conducted sessions in 4 hours to answer the third research question. This multinational study was conducted in two different countries, providing a broad perspective on the performance of these notations. Our multinational study divided the participants into three groups: one using block-based notation, another using text-based notation, and the third using both. After training, the participants were assigned practical programming tasks.

This paper is an extended version of our previously published conference paper [22]. Although the core findings remain consistent, we have introduced substantial new content, including a clearer articulation of the motivation and explicitly defined research questions. Our multiple-representation environment, Poliglot, is now introduced more thoroughly, with additional usage examples to illustrate its capabilities. The experimental design and procedure are newly presented. In addition, a background study was added to the results, and the experiment has been repeated to obtain a larger sample size of participants.

The paper is structured as follows. The second section provides crucial insights into the background of the three different approaches used to teach beginners to program: block-based, text-based, and both simultaneously. The following section introduces the multi-representational environment Poliglot briefly, developed by one of the universities participating in this study. The fourth section reviews related studies. The fifth section illuminates the experimental design, goals, and data collection instruments. The sixth section presents the comparative results of the experimental study. The seventh section outlines and discusses the essential findings from the empirical investigation. Subsequently, the following section exposes critical threats to the validity of the results in this study. Finally, in the last section, conclusions are drawn regarding the research outcomes.

2. Background

As stated earlier, our experiment included three approaches to teaching programming. In this section, we introduce these alternatives and explain their dynamics and the potential of their use as an introduction to the programming world. Each of the tools presented was analyzed from the aspect of comprehensiveness to novice programmers, i.e., children who had never encountered programming concepts before.

Block-based notation

Block programming allows early-age students to become familiar with basic programming concepts [23] while strengthening programming logic by applying visual components. Dedicated editors provide visualization of programming constructs and learning through play. Pure block-based notation is still often used for children who encounter programming at the earliest age. Block-based notation eliminates certain logical or semantic types of errors that occur in text editors, since blocks can be combined according to clearly defined rules in advance. By adopting the rules of combining blocks, the students learn to eliminate inevitable mistakes spontaneously, contributing to faster acquisition of text-based notation.

Scratch [24] is an editor that allows one to learn block programming notation through play. It is based on a multi-panel, single-window setup, which provides transparency and clear visibility at all times. Every change is noticeable immediately, giving the impression that the program is "alive". Scratch supports hands-on, one of the main approaches to practicing coding [25], through the ease of testing each block and learning through changes and play. Although based on a block, Scratch can represent a reasonable basis for adopting a bottom-up approach to programming, but also for introducing students to extremely fine-grained programming [26].

Alice is another tool that enables active engagement during the process of learning programming skills [27]. As the authors stated, one of the major challenges for novice students in programming is "putting the pieces together". Alice provides 3-D visualization for solving different programming problems. The animated programming environment in 3-D enables the students to research further and develop algorithms for animating objects in an even more intuitive and interesting way than 2-D environments provide. Creating methods for objects and testing them in the dynamic 3-D environment can enhance the adoption of object-oriented programming using text-based notation.

To approach young generations and activate them, not only when they are at the computer in the classrooms, other systems can also be used, such as App Inventor [11]. This tool is available on mobile devices. By using mobile devices, the students can practice programming spontaneously and intuitively, even in moments of leisure. With this, through the game and constant availability, interest in programming can be accelerated less formally [26]. Using such systems, it is feasible to induce students to practice programming spontaneously in their spare time [28].

Text-based notation

Python is a renowned text-based language, a multi-purpose programming language that can be utilized on various platforms [29]. The simple and minimalistic text-based syntax

makes Python a convenient programming language for beginners, whereas various specialized modules that can be imported contribute to the versatility of this programming language. According to the data in [30], based on the number of Google searches for the tutorials, Python surpassed Java in 2018, and has been the most popular programming language ever since. Python enforces indentation as a way of separating nested blocks of code, which leads to a more visually intuitive way of reading and understanding the code (since indentation is considered a part of the syntax, and not just a recommendation in coding style, e.g., in Java). The role of parentheses is relatively reduced compared to Java or other object-oriented languages, where parentheses have the role of code separators and proprietorship indicators. Python is a high-level programming language with low-level machine instructions hidden from the developers, thus increasing comprehension and softening the learning curve. Another notable characteristic of Python is the dynamic assignment of variable type based on the given value of the variable, and there is no need for preemptive type declaration. Another benefit of the Python programming language is the possibility of functionality extensions that can be achieved with the addition to the program of predefined packages (modules). The separation of functionalities into modules contributes to the code's overall simplicity and reduces imports of unnecessary functionalities. In this way, the user does not need to be familiar with all functionalities at once, but can instead study module by module, depending on their needs.

Text-based is the default and established approach for learning programming, and because of that, we strive to compare the learning effects and outcomes of this notation to block-based and both notations' usage. In our research, we chose Python as a representative of text-based notation.

Both notations

Block-based languages are a popular way to introduce programming and create educational programming environments. However, users eventually need to transition to text-based notation to develop more complex programs. Significant efforts have been made to aid this transition through various methods, including presenting translated versions, dual-mode, multiple-representation, and hybrid environments. Examples of these environments include tools such as Tiled Grace [16], BlockPy [31], Pencil code [17], Droplet [32], Greenfoot [33], and Poliglot [18], all designed to address this challenge.

Let us introduce some of these environments briefly. Tiled Grace [16] is a tile-based editor for the Grace programming language. Using tiles enables visualization of the code, and there is also support for text editing. Textual editing of code expands the tiles' visualization, enabling a seamless change of the work environment and an easy transition between source code and visual representation of the code [16]. Droplet [32] is a library designed to create dual-mode environments. It translates code by inserting tags into the textual code, to indicate which parts will be represented as blocks. These blocks are then displayed by extracting the code between the tags and presenting it within the block structure. Tags are added using an external parser and precedence is handled by a custom JavaScript function that inserts parentheses into the blocks. The transition back to text-based notation is done by removing the tags while keeping the parentheses intact. On the other hand, BlockPy [31] is a web-based environment with open access targeting primarily novice programmers in Data Science. It uses Python for its text-based notation,

facilitating a smooth transition from block-based to text-based programming for beginners.

Poliglot [18], the tool used in this study, uses both notations simultaneously. For our research, it is essential to compare the benefits and limitations of using text-based, block-based, and both notations simultaneously to introduce programming to beginners. A detailed description of the Poliglot tool, including its design and functionalities, is provided in the following section.

3. Poliglot

An example of multiple-representation environments is also Poliglot[3] [18], developed by the Slovenian partners in this paper. Poliglot is an educational programming environment designed for beginners who are taking their first steps in programming, or who are already familiar with block-based programming and want to move forward and learn text programming. Our experiences teaching programming as guest lecturers in primary schools inspired the development. Previously, we used block-based languages such as Scratch and App Inventor, which engage children in programming effectively. These tools allowed users to create functional games, mobile applications, or the motion of a robot more quickly. However, when the capabilities of block-based languages were exhausted, a transition to text-based languages became necessary.

This transition posed a significant challenge for novice programmers. They had to start with the basics again, taking much longer to reach the level of complexity they had achieved with block-based languages. We frequently observed that this shift led to a loss of enthusiasm among novices. Poliglot addresses this issue by introducing both notations simultaneously from the outset. It helps beginners connect each block to its textual representation, easing the transition. By presenting both notations together, Poliglot blurs the boundaries between block-based and text-based programming. As beginners gain experience, they often find that expressing themselves in text-based notation becomes easier than using blocks. This transition occurs naturally within the Poliglot environment.

An example of the usage of the Poliglot system is shown in Figure 1. In this example, a user inputs a number and the system calculates the corresponding multiplication table. The children can choose to work in either block-based or text-based notation. When using the block-based interface, the equivalent text-based code is displayed simultaneously in the tool's top-right corner. Any changes made in the block-based environment are reflected instantly in the text-based code, and vice versa.

As discussed earlier in this paper, arithmetic or logical equations often encourage beginners to switch to text-based notation naturally within Poliglot's multiple-representation environment [18]. To facilitate this transition, we often design tasks that are inconvenient and time-consuming to perform using block-based notation. Consider the block-based program in Figure 2, where the task is to calculate the number of days that have passed in the current year. In such cases, the students typically prefer text-based notation over block-based notation, as writing the code is faster than dragging and dropping the necessary arithmetic equation blocks onto the workspace.

Regarding error reporting in Poliglot, the block-based view prevents most syntax errors; however, users may still write incomplete or incorrect code, which must be reported.

[3] https://poliglot.um.si/

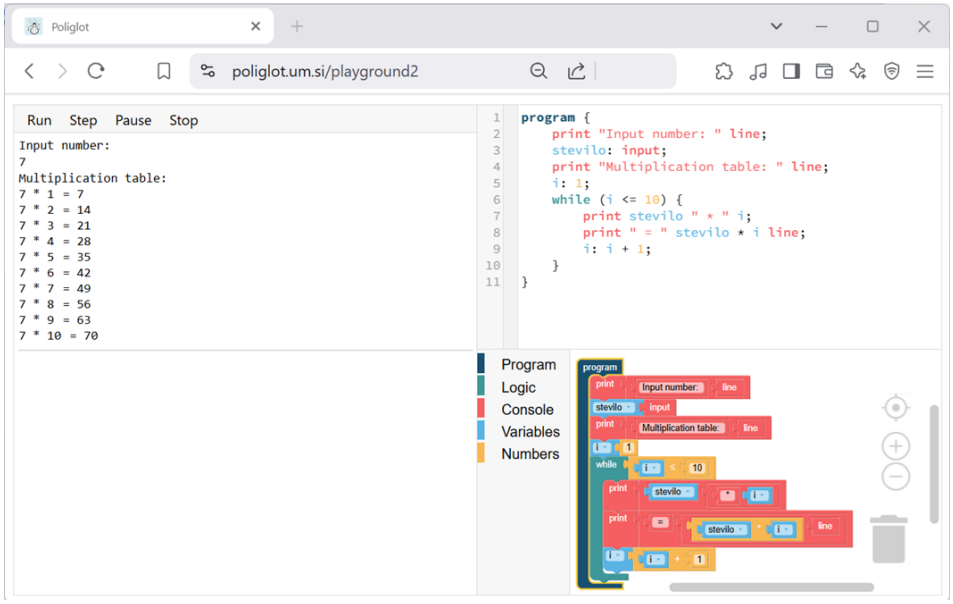

Fig. 1. Multiple-representation environment Poliglot

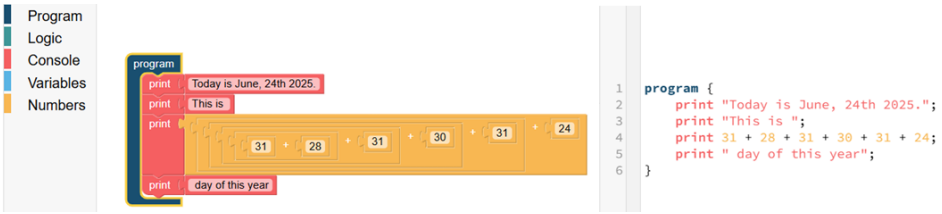

Fig. 2. Comparing arithmetic equations in block-based and text-based environments

When an error occurs within a block (such as a type mismatch in which a string is entered instead of an expected integer), the background of the text is highlighted. If the user attempts to confirm the incorrect value by pressing enter, the content reverts automatically to the last known valid state. In contrast, the textual view allows users to make any kind of error, just as in a standard text editor. Errors are detected and marked in the usual way, typically at the beginning of each row. A line containing an error will not be displayed in block-based notation.

However, once the line of code is corrected, it will appear immediately in the block-based notation, and vice versa when changing a block-based program in the text-based notation. In Poliglot simultaneous multiple representations are achieved using pretty-printing abstract syntax trees (AST), a standard task in language workbenches as described by Fowler in [34], and utilized in MPS [35]. In these programming environments, the end-user is not editing the code directly, but rather the AST, which is the model underlying the code. Programs can be understood as trees, a hierarchy of constructs that form the language behind the code. Each editor is merely one projection of the same model, and a projectional editor can have multiple projections, or representations, of the same code.

In this context, Poliglot offers two projections: a block-based editor and a text-based editor [18].

We noticed that after using both notations for a certain period, Poliglot users tend to prefer the text-based notation and recognize the advantages of text-based programming. However, we have always been interested in whether using both notations simultaneously improves their understanding of programming concepts and how it affects their programming performance.

Note that we do not favor Poliglot as a multi-representation environment. Instead, we encourage other researchers to conduct similar experiments using comparable tools, such as Grace [16], BlockPy [31], Pencil Code [17], Droplet [32], Greenfoot [33], etc.

4. Related work

The authors in [36] performed a quasi-experimental study investigating how modality (block-based and text-based environment) impacts high school Computer Science students, by conducting two classes at the same school through the same curriculum and the same teacher using either the block-based or text-based programming environment (the Pencil.cc environment was used, which supports both modalities, but the students were able to use only one modality). The outcome of this study [36] shows that the students' conceptual knowledge had been improved in both groups. However, the students using a block-based environment showed significant learning gains, as well as a higher attitude toward future programming courses. On the other hand, no difference was found in both groups with respect to confidence and enjoyment. The study [36] was extended in [8] by a third group using a hybrid block/text environment, with the main goal of how modality (block-based, text-based, hybrid block/text) influences programming practices (e.g., the number of runs, patterns in novice's help-seeking behavior). While the authors didn't find hybrid block/text modality superior, they did find some new programming practices. However, cognitive and attitudinal outcomes have not been measured or discussed. This work was later extended in [37], where the authors checked the hypothesis that gains in attitudinal and conceptual learning using a block-based environment would transfer to a conventional text-based programming language (Java). The study showed that, whilst the students had a greater conceptual learning gain using a block-based environment, this was not transferred to the environment using the professional Java programming language. Furthermore, no difference in programming practices or attitudinal shifts was found between both groups. As such, this study [37], is important, to show the limitations of block-based programming. Our study extends this one [36], and brings additional evidence in this field.

In Portugal and Serbia, a software tool for visualizing program units called C Tutor was used in introductory programming courses, and the study showed little initial association between the tool used and the test results [19]. However, the research has shown a correlation between the repeated usage of C Tutor and the achieved results via controlled practice outside the course, especially among high-achieving students. As a result, the authors have confirmed the hypothesis that "C Tutor has some positive impact on student progress in programming" [19]. The study [38] tried to answer the difference between block-based and text-based environments on novice Computer Science students' cognitive (knowledge, comprehension, application, analysis, synthesis, evaluation) and

attitudinal (satisfaction, confidence, motivation, appreciation, enthusiasm) outcomes by performing a meta-analysis, which showed that block-based environments had a small effect on cognitive outcomes, and only a trivial effect on attitudinal outcomes.

Students' difficulties in the transition from block-based to a text-based environment have been discussed in [39], where it was shown that the students struggled to solve a new coding challenge in a text-based environment due to difficulties of one or more of the following aspects: readability, memorization of commands, memorization of syntax, native language of programming, typing/spelling and writing expressions.

The authors in [40] presented a Systematic Literature Review (SLR) [41] on the characteristics of block-based environments, and how block-based environments support beginners in the transition to text-based programming, where the following distinct approaches were identified: Blocks-only, One-way Transition, Dual-modality, and Hybrid. Among those, dual-modality programming environments are the most effective for supporting students' transitioning to text-based programming.

5. Experiment design

The primary objective of this study is to examine the outcomes of assessments when beginners engage in programming using three different types of programming: block-based, text-based, and both simultaneously. By comparing the results of these approaches, the study seeks to elucidate the learning outcomes associated with teaching beginners using different programming notations.

According to the research questions defined in the introduction, the following central hypothesis is defined:

- $H1_{\text{null}}$: There are no significant differences in short-term programming performance between participants using different programming notations (block-based, text-based, or both simultaneously).
- $H1_{\text{alt}}$: There is a significant difference in short-term programming performance between participants using different programming notations (block-based, text-based, or both simultaneously).

This hypothesis will be examined from two additional perspectives that may influence the primary outcome:

- *National context*: a comparison of results between two countries: Slovenia and Serbia, and
- *Duration of programming instruction*: specifically, comparing two hours versus four hours of training.

5.1. Experiment procedure

The experiment was carried out following a clearly defined procedure (see Figure 3), consisting of several key phases, presented schematically to facilitate a better understanding of the overall process. The schematic representation illustrates the sequence of activities

within the experiment clearly, including the administration of an initial participants' background survey, the implementation of programming training with different notations, and the final assessment phase.

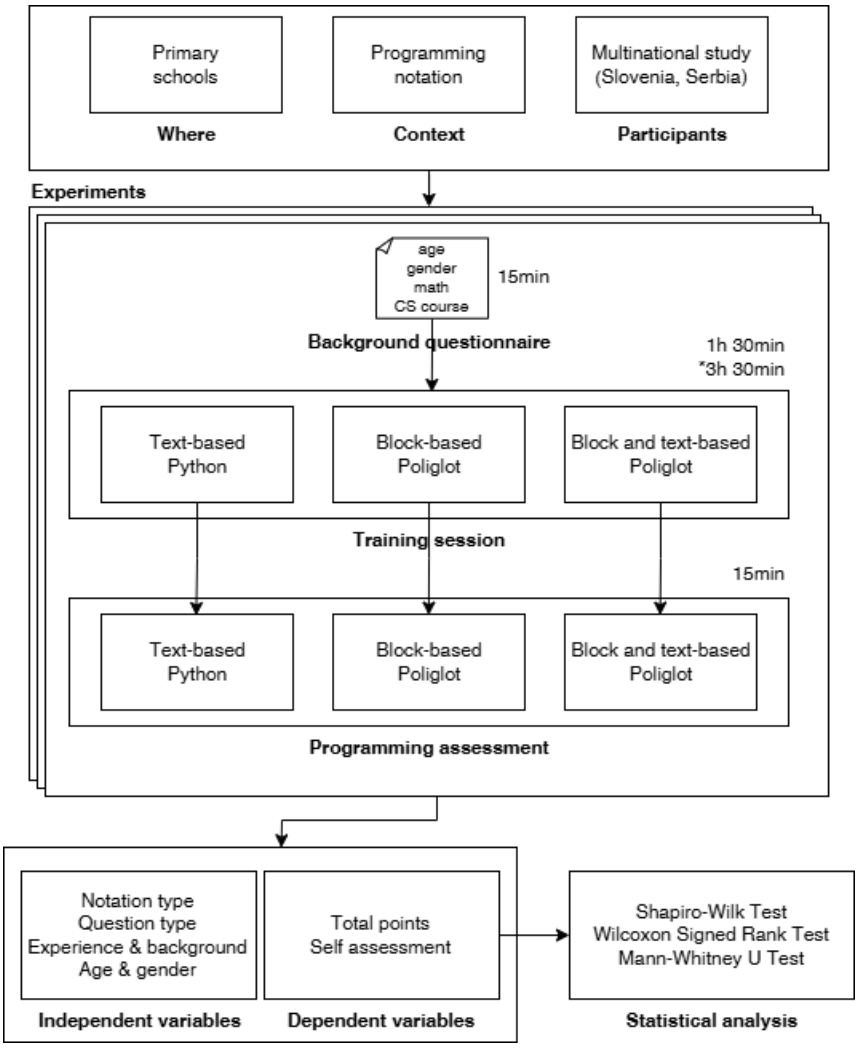

Fig. 3. Experiment procedure

The experiment was conducted in person in primary schools in Serbia and Slovenia. The participants were primary school students divided into three groups based on the notation used during the training. The first group worked with block-based notation in Poliglot, the second group used text-based notation in the Python programming language, while the third group was taught using a combination of both notations simultaneously in

Poliglot. This approach enabled a comparative analysis of the differences in understanding fundamental programming concepts among groups exposed to different instructional methods.

To ensure the consistency and validity of the results, each phase of the experiment was clearly time-defined. At the beginning of the experiment, the participants completed an initial questionnaire [4] lasting 15 minutes, which collected data on their prior knowledge and experience with computers, as well as their inclination toward solving logical and mathematical tasks. Following this, the main phase of the experiment (programming training) took place. This phase lasted 90 minutes for the participants in Serbia, while, in Slovenia, it lasted 90 minutes for one group and 210 minutes for two groups. During this phase, the participants were introduced to the theoretical foundations of programming concepts, then worked with the instructor on solving practical tasks, and, finally, solved tasks independently on computers to reinforce their acquired knowledge further.

The experiment ended with a final test lasting 15 minutes, which contained identical tasks adapted to the type of notation each group had practiced. This test evaluated the participants' programming performance. Figure 3 provides a clear representation of the sequence of these phases, offering insight into the process of conducting the experiment and the structure of activities used to investigate the impact of different types of notations on participants' success in understanding fundamental programming concepts.

5.2. Participants

The experiment was multinational and multi-institutional, involving participants from two different countries: 236 from Serbia and 155 from Slovenia. The participants were primary school children aged between 11 and 14 years. The participants were sixth, seventh, and eighth-grade primary school students. No prior participant selection was conducted for this study, resulting in the inclusion of participants with diverse backgrounds, varying levels of knowledge and experience in programming, and differing levels of interest in the subject. The assessment of previous knowledge and experience was self-assessed, supplemented by their grades the participants had at the end of a previous school year.

Multiple iterations of the study were conducted in both countries. For each new iteration, a different programming notation was used: block-based, text-based, and, finally, a combination of both notations. During the training sessions, all the educators utilized the same PowerPoint slides, which included explanations of the concepts, tasks, and correct program examples. This approach ensured consistency in the training provided by different educators. Prior to the main experiment, pilot studies were conducted, to refine the background questionnaire, training materials, and the final test. Each question in the test offered five potential answers, with only one being correct.

5.3. Data collection instruments

The tests that were handed out after the lectures consisted of the same set of questions written in the corresponding notation based on the materials that were presented during the lectures (block-based, text-based, or both simultaneously). The questions appeared in ascending order according to their difficulty. There were five question types:

[4] https://github.com/tomazkosar/PoliglotStudyOnNotations

1. Prediction of the given code execution: The participants were presented with a few lines of code with options on what the result of the execution of that code would be.
2. Finding a redundant piece of code: Based on the given code block, the participants were asked to identify a redundant line of code that did not influence the program's execution.
3. Code insertion: A block of code was presented with one line missing; the participants were expected to select the line that would complete the block of code and provide a logical solution to the problem.
4. Identification of logical errors in the code: The block of code was shown with a notice stating that there was a logical error in the code that needed to be identified.
5. Code modification: The last task required the participants to change a line of code, thus changing the result of code execution to match the description of the desired code behavior.

Even though the tests were written in different notations, the logic behind the question remained the same, without any changes to the formulation of the question except the syntax. Figure 4 shows a question from a block-based test. In this question, the participants were asked about the result of running the block-based program. This question is an example of the "Prediction of the given code execution" question type. Note that half of the questions on the test for both notations were written in the text-based notation, and the other half of the questions were written using the block-based notation.

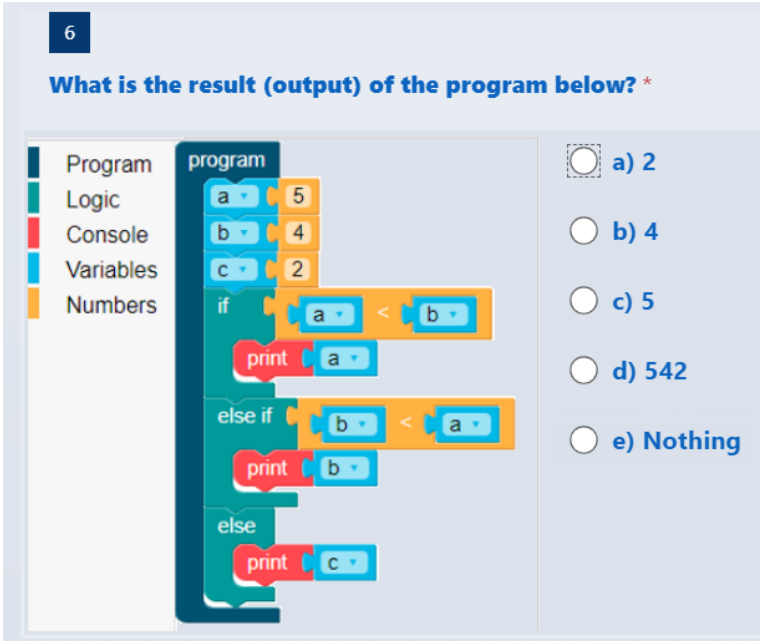

Fig. 4. Question from the block-based test

The tests were prepared in two languages, Slovenian and Serbian, allowing the participants to solve tasks in their native language. The points awarded to each question were also the same (each answer was worth 1 point) to make the questions comparable.

6. Results

This section compares the participants' performance from two different countries (Serbia is referred to as country 1 and Slovenia is referred to as country 2) with three different alternatives to teach programming. It also presents the results of the background questionnaires. All the observations were tested statistically with a α = .05 as a threshold for judging the significance.

6.1. Background study

To minimize potential bias arising from differences among participants, we compared their backgrounds across different executions, separated by country and the notation used. This comparison included previous programming experience (0 - no experience, 1 - experiences in programming), whether they had taken a mandatory or elective informatics course (0 - no informatics subject so far, 1 - had a subject in Informatics), mathematics and informatics grade (grades are between 1 and 5).

In Table 1, we present the results of a single execution, specifically from country 1 (Serbia), where we compared block-based and text-based outcomes (see Subsection 6.2).

Since the experiment followed a between-subjects design and the data did not meet the normality assumption (as determined by the Shapiro-Wilk normality test), we used the Mann-Whitney U test to compare the two groups. The results, shown in Table 1, do not indicate significant differences between the block-based and text-based groups in terms of programming experience, selection of the informatics course, or grades in math and informatics (the p-value is above 0.05 in the last column of Table 1). These findings suggest that mathematical background, programming experience, or prior informatics education are unlikely to have influenced the comparison of block-based and text-based results in Serbia.

For brevity, the results from other executions are omitted, but statistical analyses yielded similar findings. Further discussion on background results is presented in Section 7.

Table 1. Participants' background: block-based vs. text-based (Mann-Whitney U Test) – country 1

	Group	Mean	N	Std. Dev.	Median	Mean Rank	Z	p-value
Programming experience	Group I (block-based)	0.73	52	0.448	1.0	57.75	-0.477	0.634
	Group II (text-based)	0.77	65	0.425	1.0	60.00		
Mandatory Informatics	Group I (block-based)	1.00	52	0.000	1.0	59.00	-0.000	1.000
	Group II (text-based)	1.00	65	0.000	1.0	59.00		
Math grade	Group I (block-based)	3.75	52	1.118	4.0	61.32	-0.686	0.493
	Group II (text-based)	3.62	65	1.041	4.0	57.15		
Informatics grade	Group I (block-based)	4.94	52	0.308	5.0	60.69	-1.104	0.270
	Group II (text-based)	4.91	65	0.292	5.0	57.65		

6.2. Comparative analysis: two-hour executions

The following subsection presents the results of two-hour executions comparing different notations. To minimize potential biases related to cultural differences, environmental factors, and variations in primary school curricula, the results are divided by country.

Block-based vs. text-based notation results: Country 1

The first comparison examined the test results of the Serbian participants (country 1) who attended lectures and took the tests in block-based notation (Poliglot), versus those who did so in text-based notation (Python). After data cleansing, there were 52 valid responses for the block-based Poliglot test and 65 valid responses for the text-based Python test. Some tests could not be paired with their initial counterparts, and were therefore discarded. Additionally, instances of double submissions by the same participant reduced the number of tests considered for further analysis.

The Shapiro-Wilk test was used to check for normal distribution. Since the data deviated from a normal distribution, the non-parametric Mann-Whitney U test was employed to compare the two independent samples. The slight difference in mean scores (see Table 2) between the two groups was not statistically significant (p-value = 0.621). Therefore, no conclusive differences in performance outcomes could be established between the block-based and text-based groups.

Table 2. Performance results: block-based vs. text-based (Mann-Whitney U Test) – country 1

Part	Mean	N	Std. Dev.	Median	Mean Rank	Z	p-value
Group I (block-based)	45.67	52	22.40	50.00	57.29	-0.495	0.621
Group II (text-based)	48.08	65	23.72	50.00	60.37		

Block-based vs. text-based notation results: Country 2

To verify the consistency of the results obtained in Serbia a similar between-subjects study was conducted in Slovenia (country 2). Table 3 presents the performance results, measured as the percentage of correct responses, to assess the programming performance after training. Both Group I (block-based) and Group II (text-based) completed an equal number of tasks with identical question types and complexity. An examination of Table 3 reveals that the block-based group outperformed the text-based group, as indicated by the mean scores (34.19% vs. 32.92%).

Table 3. Performance results: block-based vs. text-based (Mann-Whitney U Test) – country 2

Part	Mean	N	Std. Dev.	Median	Mean Rank	Z	p-value
Group I (block-based)	34.19	39	24.47	33.33	40.73	-0.287	0.774
Group II (text-based)	32.92	40	21.84	33.33	39.29		

Once again, the data deviated from a normal distribution, necessitating the non-parametric Mann-Whitney U test to compare the two independent samples. Despite the observed difference in mean scores between the two groups, this difference was not statistically significant (p-value = 0.774). These results are consistent with our findings from the initial study conducted in Serbia – we could not confirm any statistical differences between these two groups in Slovenia.

A comparison of the results in Table 2 and Table 3 reveals that the Serbian participants performed better. This is likely attributed to differences in the participants' backgrounds, particularly the mandatory informatics curriculum in country 1. However, a direct comparison between countries is outside the scope of this study.

These results are aligned with those obtained in the study by Weintrop et al. [8] as discussed in the Related Work section.

Both notations vs. text-based notation results: Country 1

The second comparison in this research examined the use of both notations simultaneously (Poliglot) versus a text-based notation (Python). A total of 39 participants took the test using Poliglot (see Table 4). Among them, some participants did not answer any questions correctly and none achieved the maximum score. Data for the text-based environment (a total of 65 participants) were the same as shown in Table 2, but were then compared statistically with the data collected from the participants who experienced both notations simultaneously side by side during their training.

The average score on the test with both notations was the highest of all three groups, at 49.04%. The Standard Deviation for this test was with a value of 23.71%. The variance matched that of the text-based test, which was 23.72%. Once again, the difference between the groups was not statistically significant (p-value = 0.775), as determined by the Mann-Whitney U statistical test.

Table 4. Performance results: Both notations vs. text-based (Mann-Whitney U Test) – country 1

Part	Mean	N	Std. Dev.	Median	Mean Rank	Z	p-value
Group I (both notations)	49.04	39	23.71	50.00	53.83	-0.286	0.775
Group II (text-based)	48.08	65	23.72	50.00	51.85		

6.3. Comparative analysis: four-hour executions

One potential limitation that may have influenced the results of previous executions is the duration of the training sessions. To address this, in country 2 (Slovenia), we extended the training sessions to four hours for each treatment. This extension allowed the participants more time for practice, reinforcing key concepts through repetition, and enabling them to develop their solutions independently without the pressure of time constraints.

Block-based vs. text-based notation results: Country 2

Table 5 presents the performance results from the block-based and text-based notations. Both Group I (using block-based notation) and Group II (using text-based notation) demonstrated improved performance compared to the two-hour executions (e.g., see the Mean Column in Table 3). When comparing the block-based notation approach with the text-based approach, the participants achieved almost identical scores (67.26% vs 67.13%). Therefore, it was obvious that the performance difference is not statistically significant (p-value = 0.949).

Table 5. Performance results: block-based vs. text-based (Mann-Whitney U Test) – country 2

Part	Mean	N	Std. Dev.	Median	Mean Rank	Z	p-value
Group I (block-based)	67.26	21	22.53	75.00	24.64	-0.064	0.949
Group II (text-based)	67.13	27	17.73	75.00	24.39		

Both notations vs. block-based notation results: Country 2

In this experiment, we compared the performance outcomes between a group using both notations (block-based and text-based) and a group using only block-based notation. It is important to note that both treatments in this comparison were performed using the Poliglot system [18], which supports programming instruction in block-based notation alone, or in both notations simultaneously.

Table 6 presents the comparison of the performance results between the use of both notations simultaneously and the block-based notation. Both groups completed the same set of tasks. As shown in Table 6, the block-based group outperformed both notations groups, as reflected in the mean scores (67.26% vs. 60.27%). However, this difference was not statistically significant.

Table 6. Performance results: Both notations vs. block-based (Mann-Whitney U Test) – country 2

Part	Mean	N	Std. Dev.	Median	Mean Rank	Z	p-value
Group I (both notations)	60.27	28	19.56	62.50	23.16	-1.061	0.289
Group II (block-based)	67.26	21	22.53	75.00	27.45		

When comparing the block-based notation results in Tables 6 and 3, a noticeable difference can be observed in the mean scores. Since both executions were conducted under identical conditions – using the same instructors, teaching materials, and tests – we can conclude that the extended training session was the primary factor contributing to the significantly improved results in this execution. The additional time allowed for more practice and repetitions likely improved the performance of the participants.

6.4. Discovering learning patterns through association rules

In order to test the hypothesis that there are differences in performance and learning outcomes when different notations are used during the learning process, the Apriori algorithm was applied to the data retrieved from the entry tests as well as the exit test results. The Apriori algorithm is a technique for discovering hidden patterns during the data mining process. A definition of the association rule is given in [42]: association rule implies hidden association relationships that can be found among a set of objects. The algorithm has two steps: finding frequent item sets (the threshold for statistically relevant frequency is a predefined parameter - support) and generating an association rule based on the most common itemsets. The rule is derived based on the probability of occurrence of one item in the itemset if another item is present.

In this case, the algorithm was implemented in Python, leveraging the pandas library for preprocessing data, the mlxtend library for item set extraction, and matplotlib and seaborn for result visualization (heatmaps).

Before importing data to the dataframes, the data were cleaned of invalid responses, textual answers were converted to numerical values, and answers with a wide range of numbers were converted to discrete intervals, as the algorithm requires categories between which it searches for the correlations.

The first aim was to test if there was a correlation between the test type and the total number of points. The maximum number of points on the test was 8, and the minimum was 0. Based on the number of points, the rows were divided into three categories: low (0-2), medium (3-5), and high (6-8). The support was set to 0.1, as well as the minimum threshold. For country 1 (Serbia), the heatmap showed that there is no significant correlation between the score and the test type (the correlation coefficient between the category for low score and the block-based, text-based and both notations test were 0.46, 0.48 and 0.49 respectively) for the two-hour training sessions.

With the same preconditions (algorithm parameters and the duration of the training session), the data for country 2 (Slovenia) showed a strong correlation between medium performance and the use of both notations simultaneously (0.73), and a lower correlation was observed between high performance and the tests in block-based notation (0.53).

In country 2 (Slovenia), there was also an additional experiment setup where the training classes lasted for four hours instead of two. By applying the same algorithm, the results showed a strong correlation between the medium performance and the use of the both notations simultaneously test (0.73), alongside a high performance and block-based notation (0.58).

The next step was designed to test whether there is a correlation between test type and the score on the questions related to loops. The support and threshold parameters remained the same, but the category for points was reduced to low and high. For country 1 (Serbia), for classes that had a training session of two hours, there was an almost identical percentage for each test type and category (0.44, 0.57, and 0.51 for high scores in relation to block-based, text-based, and the use of both notations simultaneously tests, respectively) which leads to the conclusion that neither of the notations is better suited for learning about loops in programming.

Similarly to the first step, the same preconditions were applied during the testing in country 2 (Slovenia). The data were given as input to the algorithm with the same parameters, and the heatmap showed a strong correlation with the block-based notation and a

high score (0.71), and the use of both notations simultaneously tests and high score (0.64). Interestingly, the same results were obtained for the training sessions that lasted 4 hours.

Based on the results of the Apriori algorithm, the final conclusion for country 1 (Serbia), derived from the data collected during the two-hour training sessions, is that there is no evidence to support the existence of a correlation between the performance on the tests and the notation taught to participants. However, the results from country 2 (Slovenia) show that the participants performed better on tests that are given in both notations simultaneously and have both block and text elements.

7. Discussion

To understand the outcomes of our controlled experiment better, we present the results of a study utilizing a background questionnaire in subsection 6.1, focusing particularly on the Informatics and Mathematical backgrounds of the participants involved.

Based on the data analyzed in sections 6.1 and 6.2. two groups of between-subject tests were conducted in country 1 (Serbia). The first one was block-based vs. text-based. We cannot confirm the differences in the results statistically (see the results in Table 2, again). The fact that each participant achieved at least 1 point speaks in favor of the block-based notation, which was not the case with the text-based notation, because there were two participants with 0 points in the sample.

In favor of the uniformity of the useful prior knowledge of the participants in the two tested groups, the almost insignificant differences in the average grade in Informatics that the participants of both tested groups had (4.94 participants who did the block-based notation, 4.91 participants who did text-based notation as presented in Table 1), as well as the successfulness of resolving three logical tasks given in the input survey (the participants who did the block-based notation averaged 1.96 points from 3, while the participants who did the text-based notation had 1.97 on average). A slightly higher average grade in Mathematics was present in the participants who did the block-based notation notation (3.75) compared to the participants who did only the text-based notation (3.62). However, the average score was on the side of text-based notation, so it can be concluded that prior knowledge of Mathematics and Informatics was not crucial for the achieved result in programming in the tested groups.

The second group of between-subject tests conducted in country 1 (Serbia) was both notations simultaneously vs. text-based. Identical values of Standard Deviation and variance (see Table 4, again) in both approaches suggest the uniformity of participants within the sample, although the number of participants involved in the execution with both notations was the smallest of all three tested groups. Furthermore, the obtained results suggested a slight advantage of both notations shown simultaneously compared to the textual one, which was represented by a slightly higher average, but also by the values of the first and third quartiles, while the fourth belonged to the textual notation, because there was no maximum number of points in the execution with both notations shown simultaneously. Based on the sample, it can be concluded that the participants using block-based notation achieved better performance more easily, with lower and medium performance on the test. In contrast, for maximum performance, textual notation still had an edge. All of the above leads to the conclusion that Poliglot achieves an advantage by using both notations, and improves the average performance compared to exclusively block-based notation. Similar

to the previously analyzed group of tests (block-based vs. text-based), when comparing the groups of participants who did both notations vs. textual notation, it can be concluded that the average marks in Informatics (an average of 4.87 for the participants with both notations vs 4.94 for the participants with textual notation) and Mathematics (3.64 was the average grade for the participants learning both notations vs 3.67 for the participants learning textual notation) were quite uniform. A slight difference in the average number of points on the logical tasks of the input survey was evident in favor of the participants who did text-based notation (average 1.97 from 3 points vs average 1.77 from 3 points). Despite this, the participants with both notations achieved the best results of average points on the output of all three analyzed groups in Serbia, which can lead to the conclusion that, by applying both notations, there is scope for achieving better performance, even for those participants who are closer to block-based notation (e.g. who have experience in using Scratch or some other environments), but also among participants who are closer to textual notation, precisely because of the possibility of choosing a notation that is more convenient for them.

We evaluated our hypothesis and the results suggest that the type of programming notation (block-based, text-based, or both simultaneously) does not influence primary school participants' performance significantly in short-term introductory programming lessons. This supports the idea that educators are flexible in the decision of a programming environment to use for beginners, and this does not affect the immediate learning outcomes. The results are consistent with other similar studies [8]. Notably, even in the use of both notations simultaneously, where the participants had to use both block and text-based notation, the performance remained statistically comparable to the other groups. This implies that introducing two notations simultaneously does not affect short-term understanding, although it may impose a higher cognitive load.

Another key result highlights the positive effect of extended instructional time, regardless of the notation used. The participants who received four hours of training outperformed those who had only two hours drastically, suggesting that time investment plays a more crucial role than the choice of a notation in shaping beginner programming outcomes.

The study results were received from two participating countries, and this multinational aspect strengthens the generalizability of these results. They indicate that foundational programming skills can be taught effectively using any of the three notation types, across different cultural or educational contexts, provided that sufficient instructional time is allocated.

8. Threats to validity

This section discusses the construct, internal, and external validity threats [43] associated with our experiment.

8.1. Construct validity

In our experiment, our objective was to measure the effect of notation on test results. The participants were assessed with multiple-choice questions after two or four hours of training in specific notations: block-based, text-based, and both simultaneously. The use

of multiple-choice questions may have influenced the results. Different outcomes might have emerged if we had used code implementation instead.

The test for both notations included half of the questions in block and half in text-based notation. This may have introduced a threat to construct validity. We do not know the outcome of the test if each question had contained programs in both notations. As a result, the test may not measure the participants' conceptual knowledge accurately across both notations equally, potentially skewing the assessment results.

Another potential threat to the construct validity is the complexity of the questions. The test results in two-hour executions are generally low, with a performance average 50% or less. The outcomes of our experiments might differ if the question complexity were reduced.

The training sessions were limited to two hours, as requested by the primary schools. This constraint might have influenced our results significantly. To investigate this, we conducted executions, during which the training duration was extended to four hours. This extended training included additional functionality and repetitions of mastering the same programming concept. Consequently, we observed higher performance results (see Table 5). However, these improvements were observed in both groups, and the differences in the results were not statistically significant.

We used Poliglot for block-based and multiple-representation environments, although alternative tools exist for using both notations simultaneously (e.g., BlockPy [31]). Our experiments did not include the usage of these tools, nor did our study cover other textual languages beyond Python. Consequently, our findings are specific to the particular combination of Python and Poliglot and should not be interpreted as generalized findings, valid for all block and text-based notations. Different text-based programming languages (e.g., Java, Smalltalk, Rust, or C++) and block-based environments (e.g., Scratch, BlockPy) may yield different performance outcomes.

8.2. Internal validity

One potential threat to internal validity is the quality of instruction provided to novice programmers during our experiment. Although we standardized training materials (presentations, code snippets, etc.), the use of different lecturers may have influenced the outcomes of our experiment.

The sample size may have influenced the results of the study. To enhance the reliability and validity of these findings, it is crucial to perform multiple repetitions of the experiment with a larger sample size. This will help mitigate potential biases and provide a more robust understanding of the observed effects. Despite a relatively small sample size, the extended training period within that study demonstrated a positive influence on the results clearly.

Although all the groups received the same amount of instructional and training time, the participants using both notations were required to learn and apply two notations simultaneously in the same time slot as other groups using only the block or text-based notation. This likely imposed a higher cognitive load compared to the single-notation groups. As a result, the differences in performance outcomes between the groups may be attributed to unequal cognitive demands rather than the type of notation itself, which threatens the internal validity of the study.

The limitation of association rules analysis stems from the fact that the dataset has fewer than 400 records, and the number of participants who have taken each test is not equal. There is also a variation in the duration of the classes and the time that was dedicated to the participants practicing programming in the environment, which adds to the variation in results.

8.3. External validity

The specific context and settings of our experiment might influence our study's external validity. The results could vary with different demographics of the participants, educational environments, or levels of previous programming experience among the participants. Our findings were derived from a small set of schools in two countries. To generalize these findings, further research is needed, involving more institutions and conducting multi-institutional and multinational studies in diverse settings.

We intentionally avoided a direct comparison of student performance between the two countries. We believe that the differences in primary school curricula are too substantial to allow for a valid cross-national comparison. As previously noted in this paper, Informatics is a mandatory subject in Serbian primary schools, while, in Slovenia, it is offered as an optional subject. In Serbia, this ensures that all the students receive a foundational education in Computer Science, digital literacy, and basic programming skills. In contrast, while motivated students in Slovenia may be exposed to similar content, the coverage is less uniform and varies significantly, depending on individual school offerings and student choices. This disparity likely contributes to the substantial difference observed in the mean performance results shown in Table 2 and Table 3. An overview of receiving introductory programming in primary schools in different countries is discussed in greater detail in [4, 5].

9. Conclusion

By evaluating the impact of programming notation on learning outcomes, this paper aims to provide insights into the use of three distinct approaches for teaching programming to beginners.

In conclusion, our study examines the choice of notation, whether text-based, block-based, or both notations used simultaneously, for teaching programming concepts to novice programmers during short-term visits to primary schools. The results of our study, conducted in two countries, show that the choice of notation (whether block-based, text-based, or both simultaneously) did not affect the short-term programming performance of primary school children significantly. The results of data mining with the Apriori algorithm suggest that, for country 1 (Serbia), there is no significant correlation between performance and notation types during the two-hour sessions. The results of country 2 (Slovenia) indicate that the performance was slightly better for tests in multiple-representation environments. Even when the training duration was extended from two to four hours, the performance differences between the notation groups remained statistically insignificant. These findings suggest that educators have the flexibility to choose the programming notation for beginner sessions without impacting immediate learning outcomes negatively in terms of performance. However, the performance improved with a longer training time,

highlighting the importance of instructional duration over notation type in beginner programming education. The potential sacrifice of lower performance results for teaching both notations simultaneously in the same time slot as other notations was rejected by the results of this study. This finding suggests that introducing both notations simultaneously is a viable strategy even within time-constrained settings, as it does not compromise beginners' performance on basic concepts of programming.

For future work, longitudinal studies and follow-up research are essential, to explore the potential effects of teaching various notations in greater depth. We are planning additional experiment repetitions [44] with the same and similar settings, to validate our current findings, ensuring greater accuracy and reliability. Additionally, future research could explore whether different programming languages (e.g. C++) and tools (e.g. BlockPy [31]) offer different cognitive or performance outcomes compared to Poliglot. Future research should investigate whether such alternatives provide outcomes consistent with our results. In this study, we demonstrated how performance results were correlated with the duration of training (two hours vs. four hours). We intend to test our findings with future experiments. With a larger sample size, we can also analyze the participants' results with similar backgrounds, the same age, and the same conditions, thereby isolating the variables to be evaluated. Extending a multinational approach and considering diverse experiment settings is essential for a comprehensive understanding of using different programming notations and environments for teaching novice programmers.

Acknowledgments. This work is sponsored by the bilateral project "Programming Environments with Simultaneous Multiple Representations in Support of Early Programming Education" between Slovenia and Serbia. The Slovenian authors acknowledge the financial support from the Slovenian Research Agency (Research Core Funding No. P2-0041). This research has been supported by the Faculty of Organizational Sciences of the University of Belgrade.

References

1. Bubnič, B., Mernik, M., Kosar, T.: Exploring the predictive potential of complex problem-solving in computing education: A case study in the introductory programming course. Mathematics 12(11) (2024), https://www.mdpi.com/2227-7390/12/11/1655
2. Flannery, L.P., Silverman, B., Kazakoff, E.R., Bers, M.U., Bontá, P., Resnick, M.: Designing ScratchJr: Support for early childhood learning through computer programming. In: Proceedings of the 12th international conference on interaction design and children. pp. 1–10 (2013)
3. Dagienė, V., Jevsikova, T., Stupurienė, G., Juškevičienė, A.: Teaching computational thinking in primary schools: Worldwide trends and teachers' attitudes. Computer Science and Information Systems 19(1), 1–24 (2022)
4. Oda, M., Noborimoto, Y., Horita, T.: International trends in k-12 computer science curricula through comparative analysis: Implications for the primary curricula. International Journal of Computer Science Education in Schools 4(4), 24–58 (2021)
5. Szabo, C., Sheard, J., Luxton-Reilly, A., Simon, Becker, B.A., Ott, L.: Fifteen years of introductory programming in schools: a global overview of k-12 initiatives. In: Proceedings of the 19th Koli Calling International Conference on Computing Education Research. pp. 1–9 (2019)
6. Funke, A., Geldreich, K., Hubwieser, P.: Analysis of scratch projects of an introductory programming course for primary school students. In: 2017 IEEE global engineering education conference (EDUCON). pp. 1229–1236. IEEE (2017)

7. Mladenović, M., Žanko, Ž., Zaharija, G.: From blocks to text: Bridging programming misconceptions. Journal of educational computing research 62(5), 1302–1326 (2024)
8. Weintrop, D., Wilensky, U.: How block-based, text-based, and hybrid block/text modalities shape novice programming practices. International Journal of Child-Computer Interaction 17, 83–92 (2018), https://www.sciencedirect.com/science/article/pii/S2212868917300314
9. 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. Communications of the ACM 52(11), 60–67 (2009)
10. Videnovik, M., Vlahu-Gjorgievska, E., Trajkovik, V.: To code or not to code: Introducing coding in primary schools. Computer Applications in Engineering Education 29(5), 1132–1145 (2021)
11. Wolber, D., Abelson, H., Spertus, E., Looney, L.: App inventor. O'Reilly Media, Inc. (2011)
12. Wilson, A., Hainey, T., Connolly, T.: Evaluation of computer games developed by primary school children to gauge understanding of programming concepts. In: European Conference on Games Based Learning. p. 549. Academic Conferences InternationalLimited (2012)
13. Sáez-López, J.M., Román-González, M., Vázquez-Cano, E.: Visual programming languages integrated across the curriculum in elementary school: A two year case study using "scratch" in five schools. Computers & Education 97, 129–141 (2016), https://www.sciencedirect.com/science/article/pii/S0360131516300549
14. Tsukamoto, H., Takemura, Y., Nagumo, H., Ikeda, I., Monden, A., Matsumoto, K.: Programming education for primary school children using a textual programming language. In: 2015 IEEE Frontiers in Education Conference (FIE). pp. 1–7 (2015)
15. Moors, L., Luxton-Reilly, A., Denny, P.: Transitioning from block-based to text-based programming languages. In: 2018 International Conference on Learning and Teaching in Computing and Engineering (LaTICE). pp. 57–64. IEEE (2018)
16. Homer, M., Noble, J.: A tile-based editor for a textual programming language. In: IEEE Working Conference on Software Visualisation (VISSOFT). pp. 1–4 (2013)
17. Bau, D., Bau, D.A., Dawson, M., Pickens, C.S.: Pencil code: block code for a text world. In: Proceedings of the 14th International Conference on Interaction Design and Children. pp. 445–448 (2015)
18. Leber, Ž., Črepinek, M., Kosar, T.: Simultaneous multiple representation editing environment for primary school education. In: 2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). pp. 175–179. IEEE (2019)
19. Alves, L., Gajić, D., Rangel Henriques, P., Ivančević, V., Ivković, V., Lalić, M., Luković, I., Varanda Pereira, M.J., Popov, S., Correia Tavares, P.: C tutor usage in relation to student achievement and progress: A study of introductory programming courses in Portugal and Serbia. Computer Applications in Engineering Education 28, 1058–1071 (2020)
20. Kosar, T., Mernik, M., Carver, J.C.: Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments. Empirical software engineering 17, 276–304 (2012)
21. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in software engineering. Springer Science & Business Media (2012)
22. Kosar, T., Bjeladinović, S., Ostojić, D., Škembarević, M.S., Leber, Ž., Jejić, O.A., Furtula, F., Ljubisavljević, M.D., Luković, I.S., Mernik, M.: Teaching beginners to program: should we start with block-based, text-based, or both notations? In: 2024 19th Conference on Computer Science and Intelligence Systems (FedCSIS). pp. 395–403. IEEE (2024)
23. Zaharija, G., Mladenović, S., Boljat, I.: Introducing basic programming concepts to elementary school children. Procedia-social and behavioral sciences 106, 1576–1584 (2013)
24. Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E.: The Scratch programming language and environment. ACM Transactions on Computing Education (TOCE) 10(4), 138–144 (2010)

25. Handur, V., Kalwad, P.D., Patil, M.S., Garagad, V.G., Yeligar, Nagaratna andPattar, P., Mehta, D., Baligar, P., H., J.: Integrating class and laboratory with hands-on programming: Its benefits and challenges. In: IEEE 4th International Conference on MOOCs, Innovation and Technology in Education (MITE). pp. 163–168 (2016)
26. Salant, O.M., Armoni, M., Ben-Ari, M.: Habits of programming in Scratch. In: ITiCSE '11: Proceedings of the 16th annual joint conference on Innovation and technology in computer science education. pp. 168–172 (2011)
27. Cooper, S., Dann, W., Pausch, R.: Alice: a 3-D tool for introductory programming concepts. Journal of Computing Sciences in Colleges 15(5), 107–116 (2000)
28. Maloney, J.H., Peppler, K., Kafai, Y., Resnick, M., Rusk, N.: Programming by choice: urban youth learning programming with scratch. In: Proceedings of the 39th SIGCSE technical symposium on Computer science education. pp. 367–371 (2008)
29. Martelli, A., Ravenscroft, A.M., Holden, S., McGuire, P.: Python in a Nutshell. " O'Reilly Media, Inc." (2023)
30. PYPL: PYPL - popularity of programming language. https://pypl.github.io/PYPL.html, accessed: 22.05.2024.
31. Bart, A.C., Tibau, J., Tilevich, E., Shaffer, C.A., Kafura, D.: BlockPy: An open access data-science environment for introductory programmers. Computer 50(5), 18–26 (2017)
32. Bau, D.: Droplet, a blocks-based editor for text code. Journal of Computing Sciences in Colleges 30(6), 138–144 (2015)
33. Kölling, M.: The Greenfoot Programming Environment. ACM Transactions on Computing Education (TOCE) 10(4), 1–21 (2010)
34. Fowler, M.: Language Workbenches: The Killer-App for Domain Specific Languages? http://www.martinfowler.com (2005)
35. Voelter, M., Pech, V.: Language modularity with the MPS language workbench. In: 2012 34th International Conference on Software Engineering (ICSE). pp. 1449–1450. IEEE (2012)
36. Weintrop, D., Wilensky, U.: Comparing block-based and text-based programming in high school computer science classrooms. ACM Transactions on Computing Education (TOCE) 18(1) (oct 2017), https://doi.org/10.1145/3089799
37. Weintrop, D., Wilensky, U.: Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms. Computers & Education 142, 103646 (2019)
38. Zhen Xu, Albert D. Ritzhaupt, F.T., Umapathy, K.: Block-based versus text-based programming environments on novice student learning outcomes: a meta-analysis study. Computer Science Education 29(2-3), 177–204 (2019), https://doi.org/10.1080/08993408.2019.1565233
39. Espinal, A., Vieira, C., Guerrero-Bequis, V.: Student ability and difficulties with transfer from a block-based programming language into other programming languages: a case study in Colombia. Computer Science Education 33(4), 567–599 (2023), https://doi.org/10.1080/08993408.2022.2079867
40. Lin, Y., Weintrop, D.: The landscape of block-based programming: Characteristics of block-based environments and how they support the transition to text-based programming. Journal of Computer Languages 67, 101075 (2021), https://www.sciencedirect.com/science/article/pii/S259011842100054X
41. Kosar, T., Bohra, S., Mernik, M.: A Systematic Mapping Study driven by the margin of error. Journal of Systems and Software 144, 439–449 (2018)
42. Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R.: Advances in knowledge discovery and data mining. American Association for Artificial Intelligence (1996)
43. Feldt, R., Magazinius, A.: Validity threats in empirical software engineering research - an initial survey. In: 22nd International Conference on Software Engineering & Knowledge Engineering (SEKE'2010), Redwood City, San Francisco Bay, CA, USA, July 1 - July 3, 2010. pp. 374–379. Knowledge Systems Institute Graduate School (2010)

44. Kosar, T., Gaberc, S., Carver, J.C., Mernik, M.: Program comprehension of domain-specific and general-purpose languages: replication of a family of experiments using integrated development environments. Empirical Software Engineering 23, 2734–2763 (2018)

Tomaž Kosar is an Associate Professor at the Faculty of Electrical Engineering and Computer Science, University of Maribor. He received his M.Sc. and Ph.D. degrees in Computer Science from the University of Maribor. His research interests include programming languages, domain-specific languages, model-driven development, program comprehension, and empirical software engineering. He is a member of ACM, a reviewer for several international research journals, and a program committee member for various international conferences. He currently serves as the head of the Computer Science and Information Technologies study programmes at the University of Maribor.

Srđa Bjeladinović is an Associate Professor at Faculty of Organizational Sciences, University of Belgrade. He received his M.Sc. and Ph.D. degrees in Information Systems from University of Belgrade. His research interests are databases, information systems development methodologies, integrated software solutions and ERPs. In recent years he has been researching NoSQL and hybrid databases.

Dragana Ostojić works as a teaching assistant at the Faculty of Electrical Engineering and Computer Science, University of Maribor. She completed her BSc and MSc degrees in computer science at the University of Maribor in 2019 and 2021. Since 2019, she has been part of the Programming Methodologies Laboratory.

Milica Škembarević is a teaching assistant at the Faculty of Organizational Sciences, University of Belgrade. She completed her master's thesis on consistency in distributed information systems in 2021 at the University of Belgrade. Her research interests include databases, information systems development methodologies, data mapping, and interoperability, with a particular focus in recent years on distributed systems.

Žiga Leber is a researcher at the University of Maribor, Faculty of Electrical Engineering and Computer Science. He received his MSc degree in Computer Science from the University of Maribor. His research interests include programming languages, domain specific languages, multiple representation environments and parsing.

Olga A. Jejić is a teaching assistant at the Faculty of Organizational Sciences, University of Belgrade. She earned her master's degree in 2021 at the same university, focusing her thesis on defining software architecture modalities based on the event-sourcing architectural pattern. Her research interests include databases and methodologies for developing information systems.

Matej Moravec received his BSc and MSc degrees in computer science from the University of Maribor, Maribor, Slovenia, in 2017 and 2019, respectively. He is currently pursuing a PhD in computer science and works as a teaching assistant at the Faculty of Electrical Engineering and Computer Science, University of Maribor. Since 2019, he has

been a member of the Programming Methodologies Laboratory. His research interests include evolutionary computation and single- and multi-objective dynamic optimization.

Filip Furtula is a Teaching Assistant at the Faculty of Organizational Sciences, University of Belgrade. He holds an M.Sc. in Information Systems from the University of Belgrade. His professional interests span software engineering, programming languages, and modern application architecture. He is particularly focused on domain-driven and model-driven approaches, as well as contemporary web technologies. In the past few years, his work has been oriented toward distributed systems, exploring microservices, event-driven design, and the actor-based computational model.

Miloš Ljubisavljević is a Teaching Assistant at Faculty of Organizational Sciences, University of Belgrade. He holds an M.Sc. degree in Information Systems from the University of Belgrade. His main interests include programming languages, domain-driven design, model-driven design and web technologies. In recent years, he has been researching topics around distributed systems such as microservice architectures, actor model of computing and event-driven design.

Ivan Luković received his diploma degree (5 years) in Informatics from the Faculty of Military and Technical Sciences in Zagreb in 1990. He completed his M.Sc. (former Mr, 2 years) degree at the University of Belgrade, School of Electrical Engineering in 1993, and his Ph.D. at the University of Novi Sad, Faculty of Technical Sciences in 1996. Currently, he works as a Full Professor at the Faculty of Organizational Sciences of the University of Belgrade, where he lectures in several Computer Science and Informatics courses. His research interests are related to Database Systems, Information Systems, Business Intelligence, Software Engineering, and Data Science. He is the author or co-author of over 200 papers, 4 books, and 30 industry projects and software solutions in the area. He supervised 13 completed Ph.D. theses. He created a new set of B.Sc. and M.Sc. study programs in Information Engineering, i.e. Data Science, at the Faculty of Technical Sciences. The programs were accredited for the first time in 2015. Currently, he is a chair of Managing Board of the Computer Science and Information Systems (ComSIS) journal. He is a member of Serbian AI Society.

Marjan Mernik received the MSc and PhD degrees in Computer Science from the University of Maribor in 1994 and 1998, respectively. He is currently a professor at the University of Maribor, Faculty of Electrical Engineering and Computer Science. From 2007 to 2017, he was a visiting professor at the University of Alabama at Birmingham, Department of Computer and Information Sciences. His research interests include programming languages, domain-specific (modelling) languages, grammar and semantic inference, and evolutionary computations. He is the Editor-in-Chief of the Journal of Computer Languages, as well as an Associate Editor of the Applied Soft Computing Journal and Swarm and Evolutionary Computation Journal. He has been named a Highly Cited Researcher for the years 2017 and 2018. More information about his work is available at https://lpm.feri.um.si/en/members/mernik/.

Matej Črepinšek received his B.Sc. in 1999 and his Ph.D. in 2007, both in Computer Science from the University of Maribor, Slovenia. He is currently an Associate Professor

at the Faculty of Electrical Engineering and Computer Science, University of Maribor. His research interests include game development, mobile development, grammar inference, evolutionary computation, single- and multi-objective optimization, and computer science education.

Received: April 01, 2025; Accepted: November 30, 2025.