DOI: 10.15439/2024F448
ISSN 2300-5963 ACSIS, Vol. 39

Teaching Beginners to Program: should we start with block-based, text-based, or both notations?

Tomaž Kosar*, Srđa Bjeladinović†, Dragana Ostojić*, Milica S. Škembarević†, Žiga Leber*, Olga A. Jejić†, Filip Furtula†, Miloš D. Ljubisavljević†, Ivan S. Luković†, Marjan Mernik*

* University of Maribor, Faculty of Electrical Engineering and Computer Science, Maribor, Slovenia
{tomaz.kosar, dragan.ostojic, ziga.leber, marjan.mernik}@um.si

† University of Belgrade, Faculty of Organizational Sciences, Belgrade, Serbia
{srdja.bjeladinovic, milica.skembarevic, olga.jejic, filip.furtula, milos.ljubisavljevic, ivan.lukovic}@fon.bg.ac.rs

***Abstract*—Teaching programming poses countless challenges. One of them is determining the most effective notation to introduce coding concepts to beginners. This paper examines the merits and drawbacks of introducing block-based, text-based, or both notations at the same time when it comes to learning basic programming concepts. By comparing these approaches, the objective of this research is to clarify and assess the learning outcomes related to teaching beginners through different notations. In this empirical study, we report on a controlled experiment during short-term visits that promoted programming in primary schools. Our multinational study divided participants into three groups, one using block-based, one using text-based, and one using both notations. After training, the participants were solving practical programming assignments. The study results revealed that the participants' performance was not influenced by notation usage, as there was no statistical significance between the three groups. However, the performance outcomes were correlated with the duration of the sessions. Our findings from the controlled experiment suggest that educators can utilize different notations confidently while teaching beginners the first steps in programming.**

I. 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], creativity, communication, and teamwork capabilities. Despite its importance, programming is still not a part of the primary school curriculum in many countries (e.g., Slovenia). This gap in primary school education can lead to an essential void in the future job markets. By integrating programming into the curriculum, 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 children demonstrating software development. During these visits, we aim to engage 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 children. 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 for teaching programming to children during visits. Another critical decision involves selecting the appropriate notation for introducing programming concepts. Using block-based notation (e.g., Scratch [2], App Inventor [3]), allows for the creation of visually appealing games and applications. Conversely, text-based notation enables the teaching of fundamental programming concepts. Each approach has its advantages and disadvantages.

On the other hand, the transition from block-based to text-based programming is often highlighted as problematic; starting with block-based notation can lead to novice programmers being reluctant to switch to textual notation. This is a common pitfall that programming educators encounter during short-term visits and in the regular curriculum.

To address this problem, educational tools have emerged that enable educators to teach novice programmers using both notations simultaneously [4], [5], [6]. The most significant advantage of this multi-representational environment 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 for the expression of math equations in a more natural and efficient manner. This dual-notation strategy helps students integrate seamlessly and understand both forms of programming, easing the learning curve and enhancing their overall comprehension.

Although we developed one such multi-representational environment called Poliglot [6] that enables programming with both notations simultaneously, we are still determining if this duality impacts the mastery of basic programming concepts.

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 University of Belgrade.

Therefore, this paper's motivation is to explore which notation, block-based (Scratch, App Inventor), text-based (e.g., Python), or dual approaches (e.g., Poliglot), enhances the understanding of basic programming concepts most effectively and establishes a solid foundation for children's interest in programming. More specifically, we were motivated by the following research question: Does specific programming notation affect the participants' test performance after training sessions during short-term visits to primary schools?

We designed a controlled experiment [7], [8], [9], conducted in classrooms during short-term visits (2 hours), including training and a brief test at the end of the sessions. This multinational study was executed in two different countries, providing a broad perspective on the effectiveness 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 undergoing training, the participants were assigned practical programming tasks. The results of the study indicated that the type of notation used did not affect the participants' performance.

The paper is structured as follows: The second section provides crucial insights into the background of the three different notations/environments: block-based, text-based, and multi-representational environments (block-based and text-based notation presented 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.

II. Background

As stated earlier, our experiment included three notations. In this section we introduce these alternatives, and explain their dynamics and potential for their usage as an introduction to the programming world. Each of the presented tools was analyzed from the aspect of comprehensiveness to the 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 the basic programming concepts while strengthening programming logic by applying visual components. Dedicated editors provide visualization of programming constructs and learning through play. Pure block notation is still often used for children who encounter programming at the earliest age. Block 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 textual notation.

Scratch is an editor that allows learning 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 immediately noticeable, giving the impression that the program is "alive" [10]. Scratch supports hands-on, one of the main approaches to practicing coding [11], 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 in programming, but also for introducing students to extremely fine-grained programming [12].

Alice is another tool that enables active student engagement during the process of learning programming skills [13]. 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 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 textual notation.

To approach the young generations and activate them, not only when they are at the computer in the classrooms, but other systems can also be used, such as App Inventor. This tool is available on mobile devices. By using mobile devices, 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 [12].

Text-based notation: Python is renowned as a multi-purpose programming language that can be utilized on various platforms [14]. 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 [15], 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 of predefined packages (modules) to the program. 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.

Both notations - Multiple-Representation Environments: Block-based languages are a popular way to introduce programming and create educational programming environments. However, users eventually need to transition to textual 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 such tools as Tiled Grace [4], BlockPy [16], Pencil code [5], Droplet [17], Greenfoot [18], and Poliglot [6], all designed to address this challenge.

Let's introduce some of these environments briefly. Tiled Grace [4] is a tiled-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 tiles' visualization, enabling seamless change of the work environment and easy transition between source code and visual representation of the code [4]. Droplet [17] 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 [16] is a multiple-representation, 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.

III. Poliglot

An example of multiple-representation environments is also Poliglot[1] [6], 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. Our experiences teaching programming as guest lecturers in primary schools inspired the development. Previously, we often started with block-based languages like Scratch and App Inventor, which engage children in programming effectively. These tools allowed users to create functional games, mobile applications, and more quickly. However, when the capabilities of block-based languages were exhausted, transitioning to text-based languages became necessary.

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

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 learners.

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 learners gain experience, they often find that expressing themselves in text-based notation becomes easier than using blocks. This transition happens naturally within the Poliglot environment.

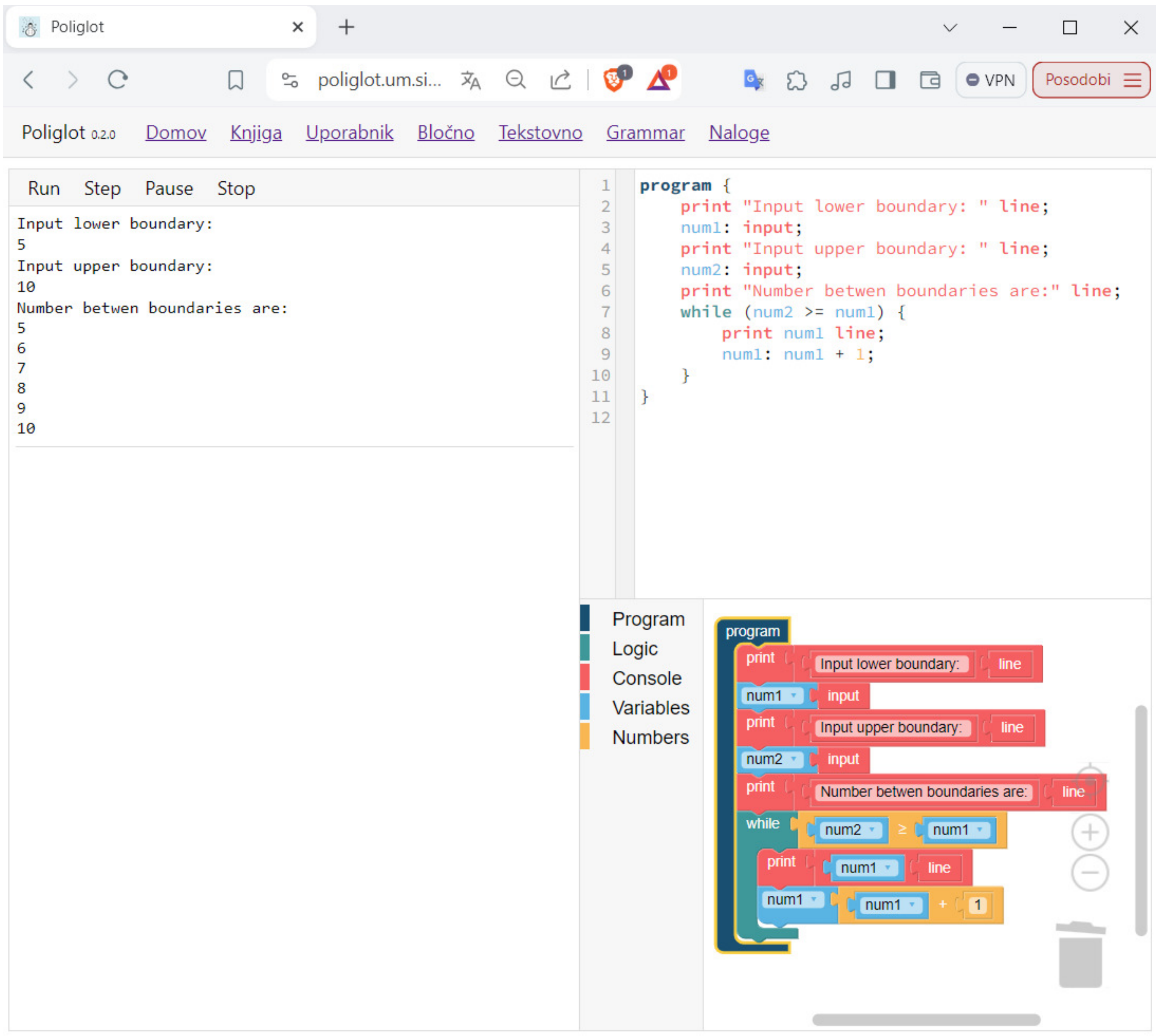
An example of the Poliglot system is shown in Figure 1. In this example, users input two numbers. Children can choose to use either block-based or text-based notation. When working in block-based notation, the corresponding text-based code appears simultaneously in the tool's top-right corner. User input in the block-based environment results in real-time text-based code updates, and vice versa. As noted, arithmetic or logical equations often prompt beginners to switch to text-based notation naturally within the multiple-representation environment Poliglot [6].

Poliglot employs pretty-printing abstract syntax trees (AST), a standard task in language workbenches as described by Fowler in [19], and also utilized in MPS [20]. 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 in MPS 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 [6].

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 [4], BlockPy [16], Pencil Code [5], Droplet [17], Greenfoot [18], etc. This will help to strengthen the results from this study.

IV. Related work

The authors in [21] 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 students were able to use only one modality). The outcome of this study [21] 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



```
Input lower boundary:
5
Input upper boundary:
10
Number betwen boundaries are:
5
6
7
8
9
10
```

```
program {
    print "Input lower boundary: " line;
    num1: input;
    print "Input upper boundary: " line;
    num2: input;
    print "Number betwen boundaries are:" line;
    while (num2 >= num1) {
        print num1 line;
        num1: num1 + 1;
    }
}
```

Fig. 1. Multiple-representation environment Poliglot with Limpid language

courses. On the other hand, no difference was found in both groups with respect to confidence and enjoyment. This work was later extended in [22], where the authors checked a 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 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 [22], is important to show the limitations of block-based programming.

The study [23] 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.

The study [21] was extended in [24] 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 and discussed. Our study extends this one [21], and brings additional evidence in this field.

Students‘ difficulties in the transition from block-based to text-based environment have been discussed in [25], 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 following aspects: readability, memorization of commands, memorization of syntax, native language of programming, typing/spelling and writing expressions.

The authors in [26] presented a Systematic Literature Review (SLR) [27] on the characteristics of block-based environments, and how block-based environments support learners 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.

V. Experiment design

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

The central hypothesis is that the type of notation used affects the participants' performance significantly. This hypothesis is based on the widespread use of block-based notation among educators teaching children the first steps in programming.

A. Participants

The experiment was multinational and multi-institutional, involving participants from two different countries: 236 from Serbia and 64 from Slovenia. The participants were primary school children aged between 11 and 14 years. The participants were sixth, seventh, and eighth-grade elementary school students. There were 84 sixth-grade students, 96 seventh-grade students, and 56 eighth-grade students from Serbia. In Slovenia, the participants were also elementary school students from the sixth to eighth grade. There were 30 sixth-grade, 29 seventh-grade, and four eighth-grade 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 based on the grades participants had at the end of a previous school year. The average grade for mathematics was 3.8 out of 5, and 4.9 out of 5 for informatics but only 162 participants had that subject.

B. Procedure

Each execution consisted of a background questionnaire, a programming class, and a final programming test, all within a two-hour timeframe. The ideal duration of the experiment would be four hours, during which the candidates would have more time to try and experiment with the tool themselves, but the duration of the experiment was limited because of practical reasons; according to the schools' schedules Informatics classes last for 1 hour, and even merging two classes was causing inconvenience to the teachers. The background questionnaire assessed the participants' prior programming experience. Before the lectures, the participants were all given the same entrance questionnaire. Apart from the elementary questions, such as grade and gender, additional data about participants were gathered through various categories:

1) Previous experience with programming (formal and informal);
2) Computer interaction frequency;
3) Inclination to problem-solving tasks;
4) Mathematical knowledge, and
5) Level of logical thinking applied to solving problems.

Following the survey, the test group attended a lecture featuring a presentation that provided training through a specific tool and notation. The lecture focused on fundamental programming concepts, such as statements (e.g., printing), logic, arithmetic equations, variables, conditional statements, and loops. The participants followed primarily the educator's actions displayed on a large screen. After learning the basic programming concepts, the participants completed a final test comprising six questions that covered the topics taught during the training. All the questions used the same programming notation as the training session. Even though the questionnaires were anonymous, the results collected at the end of the lectures were paired with the data collected from the entry questionnaire, increasing the research coverage and exploring the inclination to programming concepts and way of thinking.

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 on the test offered five potential answers, with only one being correct.

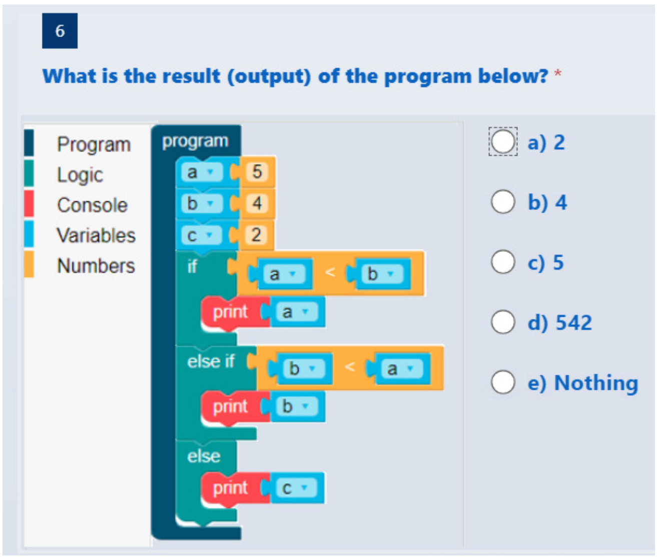
C. Data Collection Instruments

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 Poliglot, block-based Poliglot combined with text notation, or text-based Python). The questions appeared in ascending order according to their difficulty. There were five question types:

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, 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 the 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 participants to change a line of code, thus changing the result of the 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 2 shows a question from a block-based test given to the participants after training. 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.



6

What is the result (output) of the program below? *

a) 2

b) 4

c) 5

d) 542

e) Nothing

Fig. 2. Question from block-based test

Half of the questions on the test for multiple-representation environments were written in the text notation, and the other half of the questions were written using the block notation. 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 tests comparable.

VI. 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 notations.

A. Comparative Comparison: between-subjects study

Block-based vs. Text-based results (between-subjects design) - country 1

The first comparison examined the test results of the Serbian participants (country 1) who attended lectures and took the tests in block notation (Poliglot), versus those who did so in text 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 analysis further.

Statistical testing was conducted on these data, with a significance threshold set at $\alpha = .05$. 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 the mean scores (see Table I) between the two groups was not statistically significant (p-value = 0.621).

Block-based vs. text-based results: country 2

To verify the consistency of the results obtained in Serbia, another between-subjects study was conducted in Slovenia (country 2). Table II presents the performance results, measured as the percentage of correct responses, to assess programming knowledge 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 II reveals that the text-based group outperformed the block-based group, as indicated by the mean scores (36,67% vs. 34,62%).

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 = 0.831).

These results are consistent with our findings from the initial study conducted in Serbia. Similarly, they aligned with those obtained in the study by Weintrop et al. [22], which demonstrated that, when participants start programming in either block or text notation, there is no difference in correctness or efficiency when they transition to a professional text-based language. This study underscores that the initial programming notation does not impact subsequent performance.

Multiple-Representation Environments vs. Text-based results (between-subjects design) - country 1

The second comparison in this research examined a multiple-representation environment (Poliglot, featuring both text and block notation) against a text-based environment (Python). A total of 39 participants took the test using Poliglot

TABLE I
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	50,0	60,37		

TABLE II
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)	34,62	26	23,53	33,33	25,58	-0,213	0,831
Group II (text-based)	36,67	25	20,41	33,33	95,59		

(see Table III). Among them, some participants did not answer any questions correctly, and none achieved the maximum score. The data for the text-based environment (a total of 65 participants) were the same as shown in Table I, but were then compared statistically against the data collected from the participants who experienced both (block and text) notations side by side during their training.

The average score on the combined test was the highest of all three groups, at 49.04%. The Standard Deviation for this test was the same as for the text-based test, with a value of 23.71%. The variance also 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.

B. Improvement Comparison: within-subjects study

Another interesting perspective was examining how studying both notations simultaneously affected performance in a text-based notation. It is important to note that the text-based notations differed – in Poliglot, we used our text language, Limpid, while, in the text-based experiment, we used Python. This study refers to a within-subjects design, meaning that each participant first worked in a multiple-representation environment (Poliglot) and then with the text-based notation (Python).

Additionally, due to the poor initial results, we extended our study to include a 4-hour training session for each treatment. This extension provided more time for repetition, and allowed the participants ample time to develop their solutions independently without time pressure.

Table IV presents the performance results. Both Group I (using both notations) and Group II (using text-based notation) outperformed all the previous executions (e.g., see Mean Column in Table II). A closer examination revealed that the participants' performance in Python notation gave better results than Poliglot, with mean scores of 60.0% compared to 55.55%, respectively. However, the difference was not statistically significant (p-value = 0.813).

VII. DISCUSSION

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

Two groups of between-subject tests were conducted in this paper. The first one was block-based vs. text-based. We cannot confirm the differences in the results statistically. However, a higher variance of deviations indicates a greater dispersion of achieved points for textual notation. The fact that each participant achieved at least 1 point also speaks in favor of the block notation, which was not the case with the text 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 students of both tested groups had (4.98 students who did block notation, 4.93 students who did textual notation), as well as the successfulness of resolving three logical tasks given in the input survey (the students who did the block notation averaged 1.96 points from 3, while the students who did the text notation had 1.97 on average). A slightly higher average grade in Mathematics was present in the students who did the block notation (3.87) compared to the students who did only the textual notation (3.67). However, the average score was on the side of textual 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 was multiple-notation vs. text-based. Identical values of Standard Deviation and variance in both approaches suggest the uniformity of students within the sample, although the number of students with combined notation was the smallest of all three tested groups. Furthermore, the obtained results suggested a slight advantage of the combined notation compared to the textual one, which was represented by a slightly higher average grade, 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 combined one. Based on the sample, it can be concluded that the students using block 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 notation. Similar to the previously analyzed group of tests (block notation vs

TABLE III
PERFORMANCE RESULTS: MULTIPLE-REPRESENTATION ENVIRONMENTS 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		

TABLE IV
PERFORMANCE RESULTS: MULTIPLE-REPRESENTATION ENVIRONMENTS VS. TEXT-BASED (MANN-WHITNEY U TEST) – COUNTRY 2

Part	Mean	N	Std. Dev.	Median	Mean Rank	Z	p-value
Group I (both notations)	55,55	12	22,85	50,00	11,21	-0,237	0,813
Group II (text-based)	60,00	10	31,62	58,34	11,85		

textual notation), when comparing the groups of students who did combined notation vs textual notation, it can be concluded that the average marks in Informatics (an average of 4.87 for the students with combined notation vs 4.94 for the students with textual notation) and Mathematics (3.64 was the average grade for the students learning combined notation vs 3.67 for the students 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 students who did text notation (average 1.97 from 3 points vs average 1.77 from 3 points). Despite this, the students with combined notation 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 students who are closer to block notation (e.g. who have experience in using Scratch or some other environments), but also among students who are closer to textual notation, precisely because of the possibility of choosing a notation that is more convenient for them.

VIII. THREATS TO VALIDITY

This section discusses the construct, internal, and external validity threats [28] associated with our experiment.

A. Construct Validity

In our experiment, we aimed to measure the effect of notation on test results. The participants were assessed with multiple-choice questions after two hours of training in specific notations: block-based, textual, and both notations simultaneously. The use of multiple-choice questions may have influenced the results. Different outcomes might have emerged if we had used code implementation or code completion questions instead.

Another potential threat to construct validity was the complexity of the questions. The test results were generally low, with almost all the experiments resulting in an average performance of 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 one session at the university, 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 IV). However, these improvements were seen in both the multiple-representation and text-based groups, and the outcomes were not statistically significant.

We used Poliglot for block-based and multiple-representation environments, although alternative tools exist for both notations (e.g., BlockPy). Our experiments did not include the usage of these other tools, so our findings are specific to Poliglot.

B. Internal Validity

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

The sample size may have influenced the results of the within-subjects study, as only one execution was conducted, with 10 participants completing both tests. 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 the small sample size, the extended training period within that study clearly demonstrated a positive influence on the results.

Another concern for internal validity is the possibility of cheating during the tests, which could compromise the results. This is a common issue in educational settings, particularly when tests are administered in classroom environments.

C. External Validity

Our experiment's specific context and settings might influence our study's external validity. The results could vary with different participant demographics, educational environments, or levels of prior 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.

IX. Conclusion

By evaluating notation's impact on learning outcomes, this paper aims to provide insights into using three distinct approaches for teaching programming to novices.

In conclusion, our study examines the choice of notation, whether text-based, block-based, or multi-representational environments, for teaching programming concepts to novice programmers during short-term visits to primary schools. Our findings demonstrate that the choice of notation (block-based, text-based, or both) did not result in significant deviations in the participants' outcomes.

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 [29] with the same and similar settings to validate our current findings, ensuring greater accuracy and reliability. In this study, we demonstrated how performance outcomes correlated with the duration of the training (2 hours vs. 4 hours). The sample size in the 4 hour- training session was small (see Table IV, again). Therefore, 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.

References

[1] B. Bubnič, M. Mernik, and T. Kosar, "Exploring the predictive potential of complex problem-solving in computing education: A case study in the introductory programming course," *Mathematics*, vol. 12, no. 11, 2024. [Online]. Available: https://www.mdpi.com/2227-7390/12/11/1655

[2] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman *et al.*, "Scratch: programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[3] D. Wolber, H. Abelson, E. Spertus, and L. Looney, *App inventor*. O'Reilly Media, Inc., 2011.

[4] M. Homer and J. Noble, "A tile-based editor for a textual programming language," in *IEEE Working Conference on Software Visualisation (VISSOFT)*, 2013, pp. 1–4.

[5] D. Bau, D. A. Bau, M. Dawson, and C. S. Pickens, "Pencil code: block code for a text world," in *Proceedings of the 14th International Conference on Interaction Design and Children*, 2015, pp. 445–448.

[6] Ž. Leber, M. Črepinek, and T. Kosar, "Simultaneous multiple representation editing environment for primary school education," in *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2019, pp. 175–179.

[7] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[8] T. Kosar, M. Mernik, and J. C. Carver, "Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments," *Empirical software engineering*, vol. 17, pp. 276–304, 2012.

[9] L. Alves, D. Gajić, P. Rangel Henriques, V. Ivančević, V. Ivković, M. Lalić, I. Luković, M. J. Varanda Pereira, S. Popov, and P. Correia Tavares, "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*, vol. 28, no. 5, pp. 1058–1071, 2020.

[10] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The Scratch programming language and environment," *ACM Transactions on Computing Education*, vol. 10, no. 4, pp. 138–144, 2010.

[11] V. Handur, P. D. Kalwad, M. S. Patil, V. G. Garagad, P. Yeligar, Nagaratna andPattar, D. Mehta, P. Baligar, and J. H., "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)*, 2016, pp. 163–168.

[12] O. M. Salant, M. Armoni, and M. Ben-Ari, "Habits of programming in Scratch," in *ITiCSE '11: Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, 2011, pp. 168–172.

[13] S. Cooper, W. Dann, and R. Pausch, "Alice: a 3-d tool for introductory programming concepts," *Journal of Computing Sciences in Colleges*, vol. 15, no. 5, pp. 107–116, 2000.

[14] A. Martelli, A. M. Ravenscroft, S. Holden, and P. McGuire, *Python in a Nutshell*. " O'Reilly Media, Inc.", 2023.

[15] PYPL, "PYPL - popularity of programming language," https://pypl.github.io/PYPL.html, accessed: 22.05.2024.

[16] A. C. Bart, J. Tibau, E. Tilevich, C. A. Shaffer, and D. Kafura, "BlockPy: An open access data-science environment for introductory programmers," *Computer*, vol. 50, no. 5, pp. 18–26, 2017.

[17] D. Bau, "Droplet, a blocks-based editor for text code," *Journal of Computing Sciences in Colleges*, vol. 30, no. 6, pp. 138–144, 2015.

[18] M. Kölling, "The Greenfoot Programming Environment," *ACM Transactions on Computing Education (TOCE)*, vol. 10, no. 4, pp. 1–21, 2010.

[19] M. Fowler, "Language Workbenches: The Killer-App for Domain Specific Languages? `http://www.martinfowler.com`," 2005.

[20] M. Voelter and V. Pech, "Language modularity with the MPS language workbench," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 1449–1450.

[21] D. Weintrop and U. Wilensky, "Comparing block-based and text-based programming in high school computer science classrooms," *ACM Trans. Comput. Educ.*, vol. 18, no. 1, oct 2017. [Online]. Available: https://doi.org/10.1145/3089799

[22] D. Weintrop and U. Wilensky, "Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms," *Computers & Education*, vol. 142, p. 103646, 2019.

[23] F. T. Zhen Xu, Albert D. Ritzhaupt and K. Umapathy, "Block-based versus text-based programming environments on novice student learning outcomes: a meta-analysis study," *Computer Science Education*, vol. 29, no. 2-3, pp. 177–204, 2019. [Online]. Available: https://doi.org/10.1080/08993408.2019.1565233

[24] D. Weintrop and U. Wilensky, "How block-based, text-based, and hybrid block/text modalities shape novice programming practices," *International Journal of Child-Computer Interaction*, vol. 17, pp. 83–92, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212868917300314

[25] C. V. Alejandro Espinal and V. Guerrero-Bequis, "Student ability and difficulties with transfer from a block-based programming language into other programming languages: a case study in Colombia," *Computer Science Education*, vol. 33, no. 4, pp. 567–599, 2023. [Online]. Available: https://doi.org/10.1080/08993408.2022.2079867

[26] Y. Lin and D. Weintrop, "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*, vol. 67, p. 101075, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S259011842100054X

[27] T. Kosar, S. Bohra, and M. Mernik, "A Systematic Mapping Study driven by the margin of error," *Journal of Systems and Software*, vol. 144, pp. 439–449, 2018.

[28] R. Feldt and A. Magazinius, "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*. Knowledge Systems Institute Graduate School, 2010, pp. 374–379.

[29] T. Kosar, S. Gaberc, J. C. Carver, and M. Mernik, "Program comprehension of domain-specific and general-purpose languages: replication of a family of experiments using integrated development environments," *Empirical Software Engineering*, vol. 23, pp. 2734–2763, 2018.