ISSN 2676-9425

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

Volume 2

Number 2

2020

Eötvös Loránd University
Faculty of Informatics
Budapest, Hungary

Contents

A Discussion of Developing a Programming Education Portal

BALLA Tamás, KIRÁLY Sándor

Abstract. The dynamically developing IT industry is struggling with the lack of qualified software developers. The number of students in this area could be increased if more youngsters were familiar with programming in the public education system. The developed *kodosulosuli.hu* portal offers interactive programming courses and coding challenges in four different programming languages: C++, C#, Java, and Python for free. Our goal with the design and development of *kodolosuli.hu* was to give the younger generation the opportunity to learn the basics of programming in a playful way. We started operating the portal in 2015. In this paper, we primarily present the experience gained over the last 5 years and the results of the analysis of the data collected during the operation of the portal. Based on the collected experiences, both the completed improvements and the possible future development directions are presented. This paper also outlines our future goals, which are primarily aimed at enhancing learning efficiency.

Keywords: educational games, adaptive learning, computer programming

1. Introduction

In IT education, the acquisition of algorithmization skills and the development of problem-solving thinking are becoming increasingly important. If we look at the current (2020) National Core Curriculum, it can be seen that this goal, this aspiration, is clearly emphasised. The National Core Curriculum defines the educational goal of the field of informatics ("Digital culture") as "algorithmization and coding play an important role in the course, as it promotes the development of competencies such as problem solving in a digital environment, creativity, collaboration and logical thinking." (Magyar Közlöny, 31 January 2020 No. 17: National Core Curriculum p. 140.) In the last few years, achieving this goal was extremely difficult due to the lack of sufficient lessons (45 minutes once a week). This problem has proved all the more acute, since there are many programmers missing from the job market, and it would be beneficial if secondary schools were to orient students towards the software development profession. To do this, it is essential to familiarize students with programming before applying to higher education institutions.

Recognizing this problem, we set ourselves the goal of creating courses where students can acquire programming skills online in the most popular (C++, C, Java, Python) programming languages. We examined the LMS systems that are available free of charge in order to determine which one to use during the preparation of the courses. The most important goal during the course development was that we would be able to check the source codes submitted by students in an automated way. We found that there is no appropriate, small and compact LMS framework in place to check practice-oriented tasks in all three languages. Therefore, we decided to build our own framework, which provides the above possibility, and it is definitely advantageous to implement our own software system due to the easier implementation of future improvements.

As a result, we created the *kodolosuli.hu* portal, which provides online courses in C#, C++, Java and Python programming languages for free (after registration). We have been operating the portal for about five years, and the aim of our article is to present the experiences, the completed improvements and to formulate future development directions. During the development of the portal, one of our goals was to create an e-learning environment that maintains a high level of student activity, student engagement, which enables guided discovery [3],[6],[10],[16],[17],[23][26]. Our goal during the development of the portal as well as the courses was to place a strong emphasis

on the involvement of playful applications in the courses, which increase motivation and help faster and easier processing and understanding, thus making the learning process more efficient [1],[4],[19],[20],[21],[28].

In our portal, the courses are gamified. They contain game elements in a non-game context: points, incentive and immediate feedback, and after completing the different sections, users get information and screenshots about the downloadable, reward program. This software becomes available for students after completing the course. We have not implemented leaderboards, since in our experience, it motivates the best N students, and others are more frustrated by leaderboards. We have also developed 19 games to encourage learning of programming.

There are numerous games types that can enhance the effectiveness of a programming course. Games that use some kind of computing machinery (e.g., personal computer or a smartphone) are called **digital games**. A **serious game** is a digital game created with the intention of entertaining and achieving at least one additional goal, for example learning or health [5]. According to the aforementioned definitions, our developed game programs are serious games with a developed didactic purpose. Unlike serious games, **educational** or **didactic games** are definitely educating tools serving a didactic purpose. They can be defined as interactive, competitive lessons with defined learning outcomes that enable students to have fun during knowledge acquisition. Their goal is not merely fun, but they also contain an educational component [2]. Playing with some of our programs in the portal, students can simulate the working of computer programming code structures following the given instructions that help them to understand the working of codes. Following the definition [25], these games are not only education games but also **Instructional Simulation Games** as shown in Figure 1. Learners using textboxes, buttons, and slide bars can run or stop the experiment, which is a simplified version of reality, and change the parameters of the phenomena within a framework of rules.

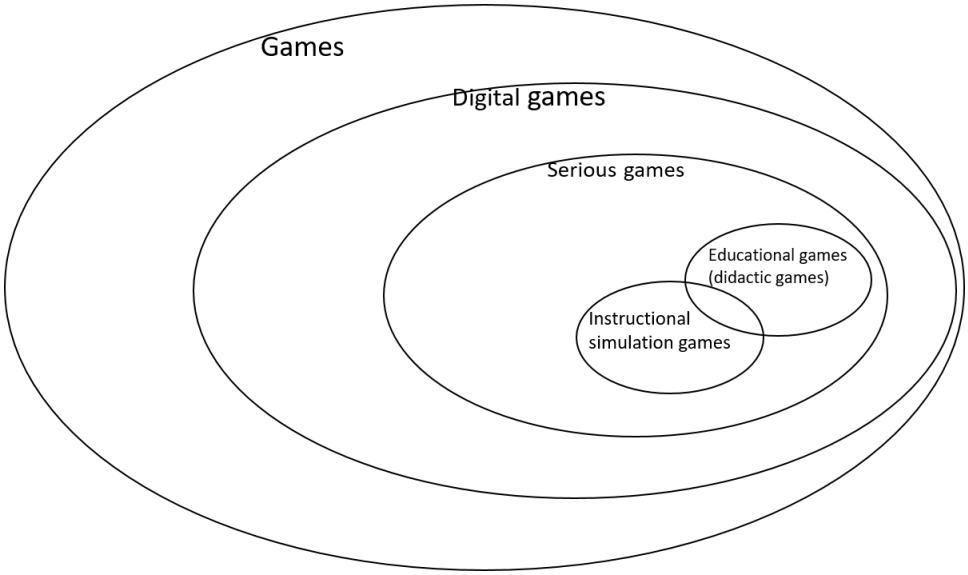


Figure 1: Game types

Our site offers real coding tasks related to the current lessons, contains animations and social media access (Facebook and Twitter).

2. The structure of the portal

The kodolosuli.hu portal offers free, online, C++, C#, Java, and Python programming languages courses in Hungarian. During the implementation of the framework, the LMS system was designed so that the courses could be structured as a series of chapters that consist of a series of lessons. Depending on the type, the lessons may include tasks as well as game programs, the purpose of which is to practice and deepen the acquired knowledge.

The website was written in PHP5 according to the MVC pattern. The curriculum, the users' data and their acts are stored in a MYSQL database in 17 tables as opposed to the game applications that can be found in folders. Currently, there are two types of users who can access to the System: students and the administrators. Students can access only the curriculum unlike the administrator who has full access to both the materials and the users' data.

After login, students need to choose a language they wish to learn. The material to be learned can be found on the left side of the screen. The exercise belonging to the current topic is on the right side (see Figure 2).

Figure 2: The main user interface of the portal. The curriculum is on the left side, the exercise and the code panel are on the right side

The aim of the chapters is to integrate the logically related knowledge elements and lessons into one unit, as shown in Figure 3. The courses for the four programming languages are structured in such a way that by completing the course, students are thoroughly prepared from basic types of process-oriented programming. The most important chapters in the curriculum are *basic types, variables, I/O operations, operators, branches, loops, complex data structures (array, list, structure, class), methods,*

text file management, exceptions. The programming theorems, which play a key role in programming education, have been inserted into the individual lessons starting from the beginning of the chapter of Loops [24],[29].

Programozási nyelvek ⇨ Java

Fejezetek listája

Sorszám	Cím	Fejezet leírása	Alfejezetek száma	Alfejezetek listája	Fejezet feldolgozása
1	Változók	Változók használata.	11	Alfejezetek	Feldolgozás
2	I/O műveletek	Hogyan lehet adatok beolvasni konzolról? És hogyan lehet a képernyőre írni adatokat?	6	Alfejezetek	Feldolgozás
3	Operátorok	A programozási nyelv által biztosított operátorok és használati esetek.	7	Alfejezetek	Feldolgozás
4	Válaszút elé érkeztünk!	Eddig azt tapasztaltuk, hogy a program utasításai **egymás után kerülnek** végrehajtásra, azaz a végrehajtás, futtatás **szekvenciális**. Hogyan lehet azt megoldani, hogy például egy változó értékétől tegyük függővé utasítás vagy utasítások végrehajtását? Erről szól a fejezet.	3	Alfejezetek	Feldolgozás
5	Ciklusok	Az előző lecke után világos, hogyan lehet bizonyos utasításokat végrehajtani vagy nem végrehajtani egy feltételtől függően. A kérdés most az, hogyan lehetne bizonyos utasításokat **többször is végrehajtani** egy adott feltételtől függően.	9	Alfejezetek	Feldolgozás

Figure 3: List of chapters (Java)

A linear sequence of lessons forms a chapter. We distinguish three types of lessons: **curriculum with assignments**, **independent assignment**, **independent curriculum**. Independent teaching materials may contain blocks of text, audiovisual elements to support the learning process, and games may also be included here.

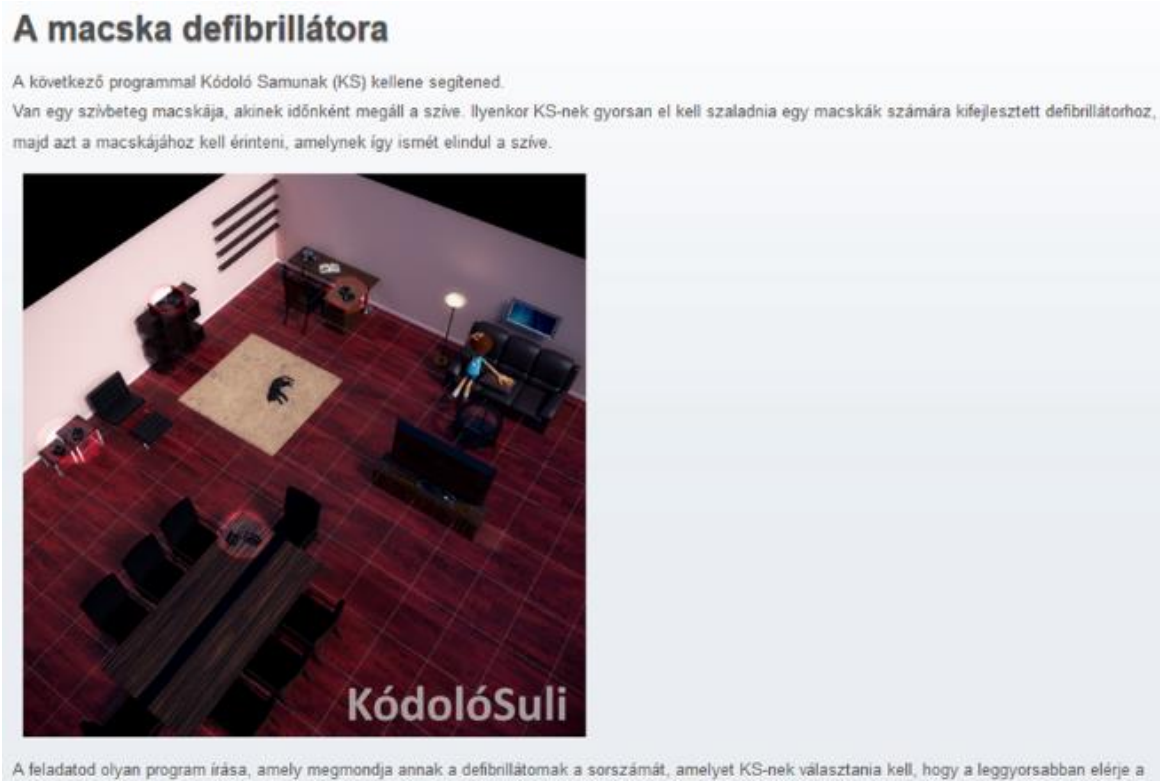


Figure 4: An independent task with a picture that depicts the problem

It is possible to create lessons with short tasks, in which case, in addition to the curriculum, short sets of tasks appear on the right side to help students master the specific part of the lesson (see Figure 2). For most of the lessons, we have attached more than one practice.

The third type of task is the independent task, in which case we expect the solution of a complex practical task (see Figure 4). The learner solves the task with the help of the development environment, uploads the source code to the server, which translates it, and decides whether the solution is correct based on the output answers given to the input defined by the program (see Figure 5). The programs are run by the system, so we considered it important that no matter how simple the tasks are, the programs should run in a sandbox similar to the one used on the *progcheck.nejanet.hu* portal [15]. Accordingly, a run time limit of 1 second is set for all programs. Memory and hard disk cannot be accessed directly by programs, and programs that try to do so will not be run.

Figure 5: An independent task after grading the uploaded solution

3. Improving the effectiveness of courses

At the beginning, our courses were available at *kodolosuli.nejanet.hu*. Secondary school students tested the portal at this stage in 2014. Most of them completed the courses and gave feedback that enabled us to correct pitfalls.

In an online environment, students' performance can be influenced by several factors that stem from students' individual habits and characteristics. These include the ability of students to maintain their attention, their intrinsic motivation to learn [6],[12]. It is also possible to influence these factors in an online environment, as we have the opportunity to increase engagement, which can be defined as a student's cognitive process, active and emotional participation in the learning process [4],[20],[28]. All three factors of commitment (cognitive, behavioral and emotional) can be increased if the text of the curriculum is not only professionally correct, but also good in style and personal in tone [7]. The thought-provoking effect of the text can be facilitated by the inclusion of interesting examples, good metaphors, sometimes astonishing and provocative recordings, questions, contradictions, surprising and in many cases humorous twists [12],[13]. We have modified the text of the curriculum accordingly.

Then we moved to *kodolosuli.hu* and opened the courses for university students who also helped us in correcting errors in 2015. Then we made the portal available for anyone after registration.

After about one hundred registrations, we checked their interactions and submissions to find the weak spots of the courses. Since all the submissions are stored in a database, we were able to find the tasks that proved extremely difficult. The coding tasks that most students could only solve with additional help have been changed, as well as the study material belonging to that task. After implementing these improvements, users were able to solve coding tasks more efficiently which means that they were able to solve each task from fewer attempts.

The next step was that we have made our programming courses gamified. They contain game elements in a non-game context as we mentioned in the first section. The usefulness of the

electronic environment has a significant effect on student satisfaction, which is why it is worth striving to make the content applicable in real life [9]. Accordingly, exercises and coding tasks that students need to complete are applicable to real life to allow students to feel the usefulness of the portal [9]. Practical exercises, such as finding the closest defibrillator to save a life, writing a code that can control the descent of a spacecraft onto a planet, how to move an object from a given (X,Y) coordinate to another position as fast as possible or controlling the parking sensor of a car can increase the efficiency of the e-learning environment. The decision theorem should also be applied if the task is worded as follows: "Given a point, the center and radius of a circle, decide if the point is inside the circle". To make this task sound more exciting, it is phrased as "We know the range of a laser cannon and the coordinates of the incoming enemy object. Let's decide if it is worth firing with the cannon." (Figure 6).

For difficult tasks, we created videos and animations to motivate students to come up with a solution.

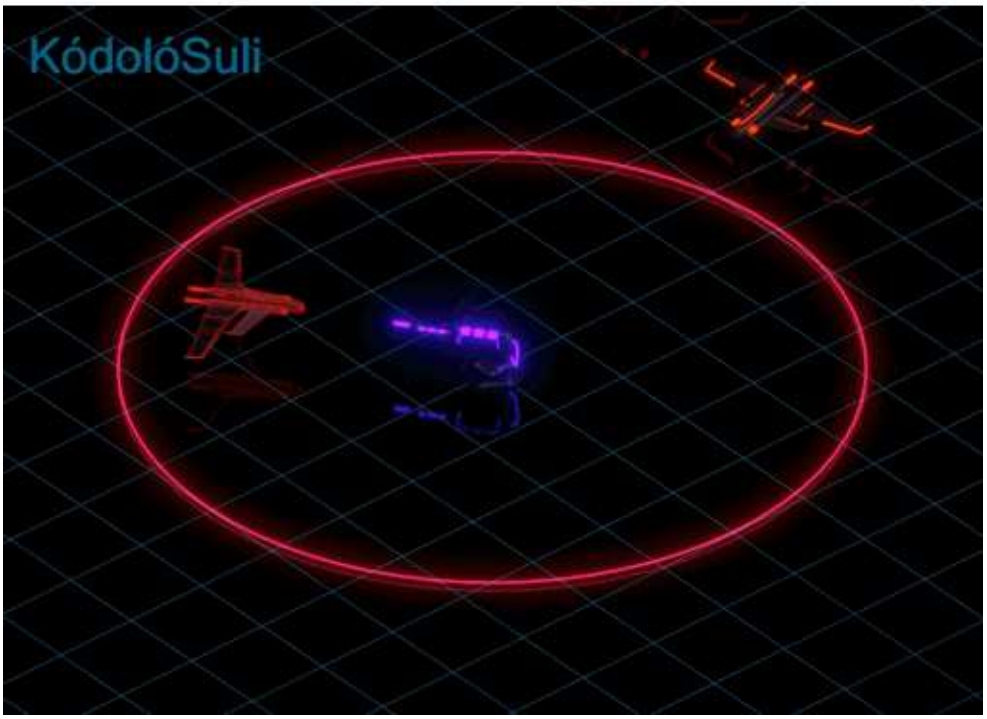


Figure 6: An animation to motivate students to come up with a solution

When creating an effective e-learning environment, the developers should strive to make the learner feel that the teacher, the instructor is present, accessible, so that the feeling of cognitive presence is realized [9]. Therefore, we have integrated the Disqus system into the portal, which is suitable for building communities. Social media is also available, most specifically FaceBook and Twitter.

In 2016, after gaining more than 400 registered users, we started to develop 19 educational game programs that cannot only foster motivation during the playing process but provide faster understanding, since students needed less attempts to solve coding tasks. Most of them are educational simulation games, but we have also implemented games that help in memorisation (memory cards, board game and hangman) and also ones that measure how well the learner has understood the material.

They were written in HTML5 and JavaScript and added to the course right before a coding task. In this paper, we introduce two of them with their learning objectives and learning activities.

3.1 The developed educational games

According to Vihavainen et al. in [27], the intervention of educational games in programming courses raises pass rates by 10.8% on average. We were curious, whether developing and adding educational games to our portal can improve its effectiveness.

3.1.1 Counting factorial by calling a function

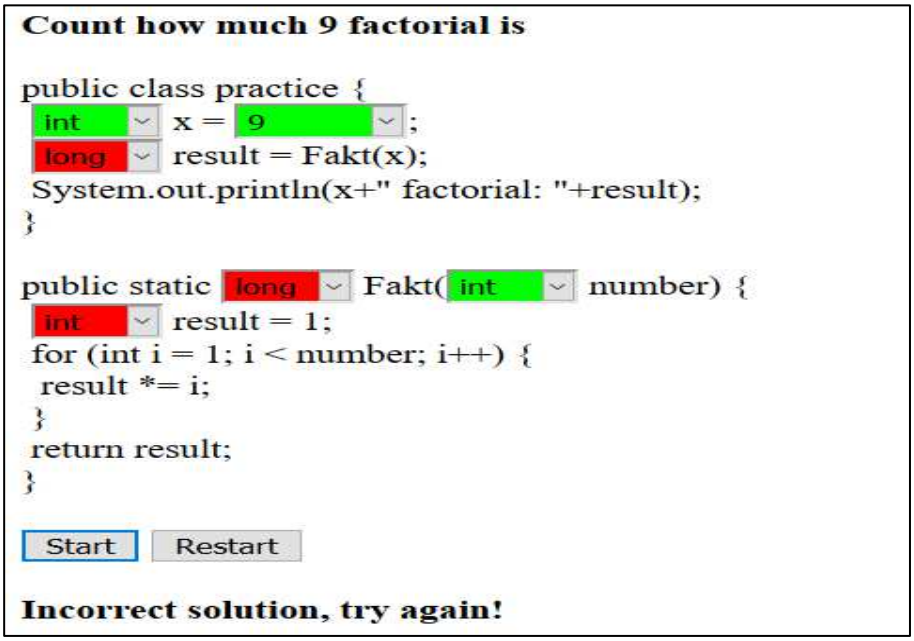


Figure 7: Writing a function with one parameter and calling it. Parameter types can be set by user

Learning goal: how to call a function with one parameter, and what happens after calling this function. In the game, the line of the programming statement to be currently executed is blue. If the parameter types are not correct, a warning message is displayed and the program cannot be executed (see Figure 7). Students can observe the flow the execution.

Activity: player has to set the value and the type of the current parameter, the type of the formal parameter and the type of the return value.

3.1.2. Throw different exceptions

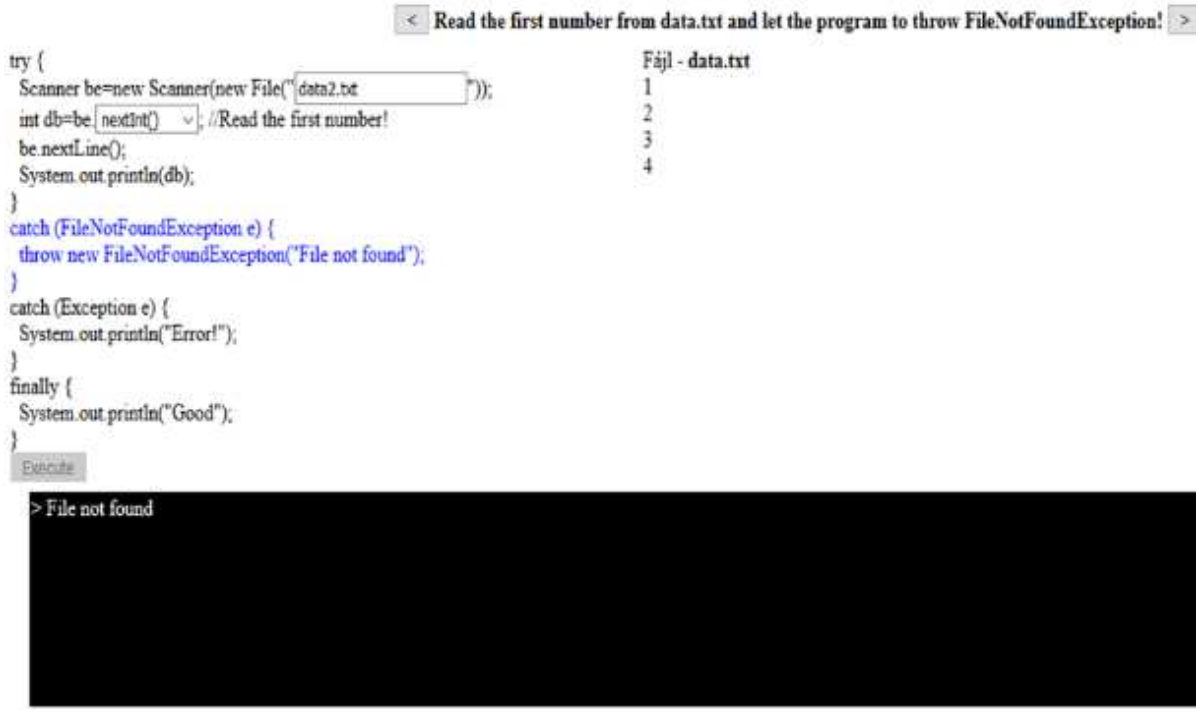


Figure 8: A game for better understanding of exceptions.

Learning goal: how to catch exceptions and how to trigger an exception. In the game, the line of the programming statement to be currently executed is blue. Students can observe the flow of the execution: what causes an exception and when, and which catch block will be executed after throwing an exception.

Activity: player has to modify statements in the code to throw the requested exception (see Figure 8).

3.2 Measuring the effect of the developed games

We have also implemented traditional board games such as Match Pairs Memory Game, Hangman and two own-developed board games, but because they were placed in the portal more than once with different contents, we could not measure their efficiency. Fifteen educational games were placed immediately before an exercise which are followed by another exercise on the current topic for which the game was created, everything else remained unchanged.

We compared the performance of the last 200 students (control group) who registered and completed the last chapter in the curriculum before the game programs had been added into the Java course, and the first 200 students (experimental group), who registered and completed the last chapter in the curriculum, after the addition of the games in the supported coding tasks. We examined how many students were able to solve the first and the second exercises after the first or second try from the 200, before adding the games to the portal (control group) and after adding them to it (experimental group).

The results were impressive and proved that with our thematic didactic game programs, users were able to solve coding tasks more effectively (with less attempts). We published the results in an other paper [14].

4. Directions for further development, opportunities

Our most important goal is to implement new, advanced courses in all four languages that move from the process-oriented world to the basics of object-oriented programming. Obviously, this involves the development of new curricula, games and tasks.

Due to the popularity of the Python language, we have added it to the list of courses, therefore it has been available since 1st September 2020. It was not possible to add all the developed games to the Python course due to the peculiarities of the Python language compared to the other three languages. It is planned to develop educational games that take into account the specifics of the Python language.

Another important goal is to examine how effective the structured curricula is, along with exercises that help to practice and deepen skills. We consider it important to implement functions in the framework with the help of which the efficiency of the curriculum and the reliability of the tests can be measured. This way, the framework also becomes suitable for building tests that really measure knowledge about the course. Here it is necessary to develop computational functions related to the topic of test theory, which can ensure the reliability, authenticity and objectivity of the tasks.

On the other hand, expanding the toolbar can also be important. At the moment, the programs implemented by the users are examined in terms of (in the case of stand-alone tasks) what output it produces for a pre-determined input. It is obviously enough for beginners to test a task at this level, but in the long run it is much more useful to build the expression tree for the source based on the language rules and subject it to preliminary analysis. In the long run, we will also able to check whether the required subroutines can be found in a program with the appropriate parameters, or in the case of new courses it is possible to determine whether the right object class with the appropriate fields and methods exists. This step can also be important for efficiency analysis, as the end goal is to achieve not only the right but also the right and effective resources. The efficiency analysis of algorithms and programs can be performed classically in three areas:

(1) Execution time is an important factor for both the user and the server programs. This is also relatively easy to measure at the moment, only the difference between the start time and the stop time of the program after compilation should be taken into account.

(2) Storage requirements, in which we examine how much memory space the program uses during its operation. This is difficult to examine in the current implementation, in the future a measurement methodology must be created in the framework.

(3) Complexity indicators, which assigns some degree of complexity to the algorithm of the implemented program. Obviously, the above expression tree construction is required for this analysis.

Finally, we also consider it very important to increase the adaptability of the system because as M. Nadas said…"in adaptability in pedagogy we mean adaptive development/evolution that takes into account the aspects of unity and differentiation, whether we approach it from the needs of individuals, groups or institutions." [18] In other words, it is a system of pedagogical activities that strives to take into account the needs of all participants in the education system at the same time.

It can be interpreted as a dynamic interaction of the concepts of change-reflection-learning/innovation [22]. To make our portal adaptive, the first step can be developing a lighter version of educational games. With these versions, students have fewer and simpler tasks during the game, and in many cases they do not have to type any code, it is enough to select the appropriate code line from the menu, as shown in Figure 9.

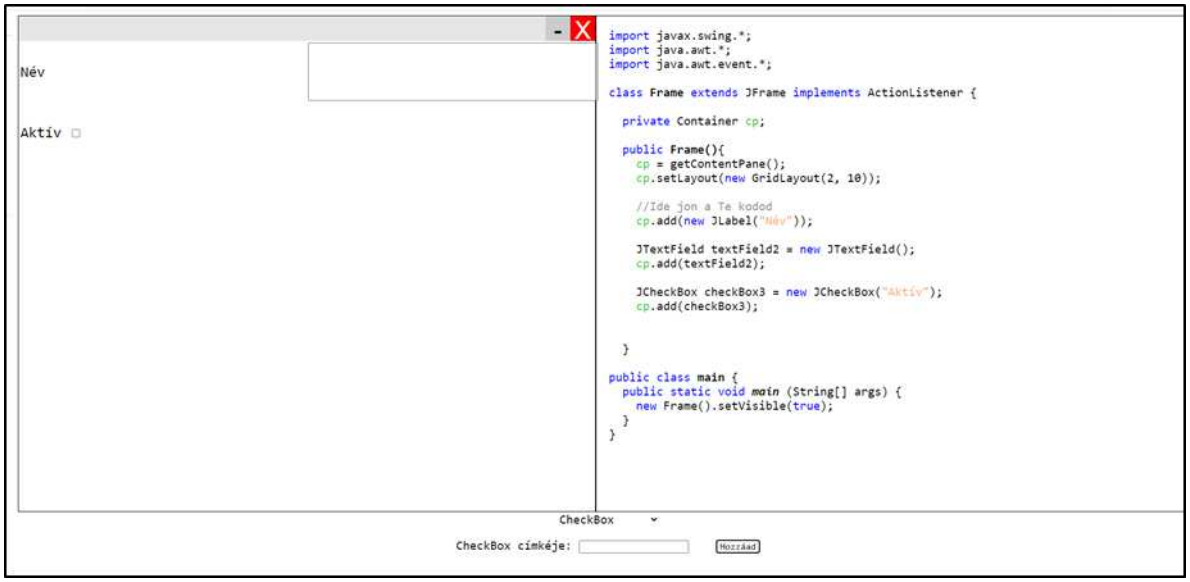


Figure 9: Frame construction by adding components. The selected component appears on the Frame and the appropriate instructions are added to the code.

We want to store the practice tasks in a question bank with a level number from 1-3 while increasing the number of tasks for each level. In the case of each lesson, students start a task at level 1, and after solving it without help, they are allowed to continue with another task at level 2. In case of failure or using help, students need to solve another task at level 1. This solution could lead to adaptive problem solving.

Achieving full adaptability is an extremely labor-intensive task, and the curriculum as well as the LMS also require major modifications. Its implementation, even when using a Moodle system, requires a lot of work from developers [11]. The first step in this work is to identify the initial parameters (age, prior knowledge, learning goals, etc.) of learners using the systems. Further measurements are needed of users' interactions with the digital environment: how much time they spent reading a lesson, solving a task, how many times they improved the solution, what order they followed, and so on. They are related to the possibility of contextual data collection and attach increasing importance to data mining technologies [8].

If we implement a learning algorithm (neural network or inference engine) that can "learn the learner's behavior", then the system is able to operate more efficiently: recognizing the important elements of the tasks, the algorithm can decide if the learner has properly learned the material. From then on, the number of tasks to be solved will be personalized. Moreover, by describing relationship networks, the direction of the step can even be determined between tasks and groups of tasks.

5. Conclusion

In the second section of this paper, we have outlined the structure and the functionality of a gamified learning portal for computer programming. We have also presented the ways and methods of improving the effectiveness of this site by using ourself-developed LMS. A well-designed LMS is suitable for tracking student activity. Since we store both the coding tasks of the course and the users' solutions in a database, it is possible to make the task solution part of the courses adaptive, which allows faster and more personalized progress in the system. Today more than 900 users have registered on the portal, who spent a significant amount of time learning the courses. The developed educational games enhanced the effectiveness of the programming course. The practical and independent tasks placed between the chapters of the courses greatly help to master the curriculum. Analysing the data, we found that there are areas in programming education that pose a challenge for learners. We will conduct further research in relation to these, and then develop solutions that will help to effectively address problems.

Bibliography [1]

1. T. Balla, S. Király: *Online programozás a kódolósuliban* (in Hungarian). In: Szlávi, Péter; Zsakó, László (Eds.) INFODIDACT 2017, Budapest, Magyarország. Webdidaktika Alapítvány, (2017) Paper: 8. https://people.inf.elte.hu/szlavi/InfoDidact17/betolt.html
2. Blumberg, F., Debby E. Almonte, Jared, S. Anthony, and Hashimoto, N: *Serious Games: What Are They? What Do They Do? Why Should We Play Them*. In: Dill, K.E. (Hrsg.): The Oxford Handbook of Media Psychology, Oxford et al. 2013. S. 334-351., (2013). DOI: 10.1093/oxfordhb/9780195398809.013.0019
3. P. de Byl, J. Hooper: *Key Attributes of Engagement in a Gamified Learning Environment*. In H. Carter, M. Gosper and J. Hedberg (Eds.), Electric Dreams. Proceedings, ASCILITE Sydney, pp.221-230., (2013).
4. R.C. Clark, R.E. Mayer: *E-learning and the science of instruction*. Pfeiffer, San Francisco, (2011)
5. Dörner, R., Göbel, S., Effelsberg, W., Wiemeyer, J.: *SeriousGames- Foundations,Concepts and Practice*. 1st ed.; Springer International Publishing: Basel, Switzerland, (2016). DOI: 10.1007/978-3-319-40612-1
6. B., Faragó: *Tanulói aktivitás, aktív tanulás és tevékenység online környezetben* (in Hungarian). In: Papp-Danka, Adrienn; Lévai, Dóra (szerk.) Interaktív oktatásinformatika, (2015).
7. K. Héjja-Nagy: *Tanulási stratégiák és a tanulói aktivitást befolyásoló egyéni feltételk online környezetben* (in Hungarian). In: Papp-Danka, Adrienn; Lévai, Dóra (szerk.) Interaktív oktatásinformatika p. 33-49., (2015).
8. L. Hülber: *Az adaptív online környezet lehetőségei az egyén fejlesztésében*. (in Hungarian) In: Papp-Danka, Adrienn; Lévai, Dóra (szerk.) Interaktív oktatásinformatika, p.138-151., (2015)
9. Joo, Y.J., JOUNG, S. & KIM, E.K.: *Structural Relationships among E-learners' sense of Presence, Usage, Flow, Satisfaction, and Persistence*. In: Educational Technology and Society, 16(2), 310-324., (2013).

[1] The data as indicated in the source are to be followed. The main principle is to make the source reachable.

10. F. Layth Khaleel, N. Sahari, T. Siti Meriam, A. Ismail: *The study of gamification application architecture for programming language course.* In: ACM IMCOM 2015 -Proceedings. Association for Computing Machinery, Inc, 2015. a17, (2015). DOI: 10.1145/2701126.2701222

11. S. Király: *Adaptív oktatás a moodle segítségével* (in Hungarian), In: Új köznevelés (2016). https://folyoiratok.oh.gov.hu/uj-kozneveles/adaptiv-oktatas-a-moodle-segitsegevel

12. S. Király: *Tanulás támogatása digitális környezetben* (in Hungarian). In: OKTATÁS-INFORMATIKA 2016: 1 pp. 29-40., 12 p., (2016)

13. S. Király*: How to Implement an E-learning Curriculum to Streamline Teaching Digital Image Processing*, In: ACTA DIDACTICA NAPOCENSIA 9: 2 pp. 13-22., 10 p., (2016)

14. S. Király, T. Balla: *The effectiveness of a fully gamified programming course after combining with serious games in a computer programming portal,* In: ACTA DIDACTICA NAPOCENSIA 13: 1 pp. 65-76., 12 p., (2020). DOI: 10.24193/adn.13.1.7

15. S. Király, Sz. Székely: *How to use our own program evaluation system to streamline teaching computer programming.* In: Teaching Mathematics and Computer Science 13:(1) pp. 73-80., (2015). DOI: 10.5485/TMCS.2015.0384

16. B. Kumar*: Gamification in education - learn computer programming with fun*. In: International Journal of Computers and Distributed Systems, Vol. No.2, Issue 1, December 2012, pp. 46-53., (2012)

17. Lengyelné, Molnár Tünde: *IKT mint oktatás támogató rendszer* (in Hungarian). In: Bárdos, Jenő; Kis-Tóth, Lajos; Racsko, Réka (szerk.) XIII. Országos Neveléstudományi Konferencia: Változó életformák - Régi és új tanulási környezetek: Absztraktkötet, (2013)

18. M. Nádas, M.: *Adaptív nevelés és oktatás* (in Hungarian). Magyar Tehetségsegítő Szervezetek Szövetsége. Budapest, (2010)

19. Muratet, M., Torguet, P., Viallet, F. & Jessel, J.-P.: *Experimental feedback on Prog&Play: a serious game for programming practice*. in: L. Kjelldahl and G. Baronosk (Eds.), EUROGRAPHICS1–8., (2010).

20. N. Pellas: *The influence of computer self-efficacy, metacognitive self-regulation and self-esteem on student engagement in online learning programs*. Evidence from the virtual world of Second Life Computers in Human Behaviour, 35, 157-170, (2014).

21. R. Racsko, L. Kis-Tóth: *Új tanulási környezetek a köznevelésben: a személyes tanulási terek és az interakcióelemzés elméleti háttere* (in Hungarian). In: Námesztovszki, Zsolt; Vinkó, Attila (szerk.) XXI. Multimédia az oktatásban és II. IKT az oktatásban konferencia = XXI Naučna konferencija „Multimediji u obrazovanju" i II Naučna konferencija „IKT u obrazovanjuˇ. Szabadka, Szerbia : Újvidéki Egyetem Magyar Tannyelvű Tanítóképző Kar, pp. 253-258. 6 p., (2015)

22. N. Rapos, K. Gaskó, O. Kálmán, Gy. Mészáros, Gy. (2011): *Az adaptív-elfogadó iskola koncepciója* (in Hungarian). Oktatáskutató és Fejlesztő Intézet. Budapest, (2011). http://mek.oszk.hu/13000/13021/13021.pdf

23. M. Sailer, J. Hense, H. Mandl, M. Klevers: *Psychological Perspectives on Motivation through Gamification, Interaction Design and Architecture(s)*. Journal - IxD&A, N.19, 2013, pp. 28-37., (2013)

24. P. Szlávi, L. Zsakó, G. Törley: *Programming Theorems Have the Same Origin*. In: Central-European Journal of New Technologies in Research, Education and Practice Volume 1: Number 1, pp. 1-12., 12 p., (2019) DOI: 10.36427/CEJNTREP.1.1.380

25. Sauvé, L., Renaud, L., Kaufman, D., & Marquis, J. S.: *Distinguishing between games and simulations: A systematic review*. Educational Technology & Society, 10 (3), 247256., (2007).

26. Tsai, M.-J., Huang, L.-J., Hou, H.-T., Hsu, C.-Y., Chiou, G.-L.: *Visual behavior, flow and achievement in game-based learning*. In: Computers & Education. 98, pp 115–129., (2016). DOI: 10.1016/j.compedu.2016.03.011

27. Vihavainen, A., Luukkainen, M. & Kurhila, J.: *Multi-faceted support for MOOC in programming*. In: Proceedings of the 13th Annual Conference on Information Technology Education, ACM, New York, pp. 171–176, (2012).

28. M. Wolf: *Learning to Think in a Digital World*. In: Bauerlein, M. (ed.): *The digital divide: arguments for and against Facebook, Google, texting, and the ages of social network*. Jeremy P. Tarcher/Penguin, New York. 34-37., (2007).

29. L., Zsakó, P. Szlávi, É. Harangozó: *Joining Programming Theorems, a Practical Approach to Program Building*. In: Annales Universitatis Scientiarum Budapestinensis de Rolando Eotvos Nominatae, Sectio Geologica 17: 1 pp. 155-172. 18 p. (1998).

Authors

BALLA Tamás

Eszterházy Károly University, Faculty of Informatics, Institute of Mathematics and Informatics, Department of Computational Science, Eger, Hungary,
e mail: balla.tamas@uni-eszterhazy.hu

KIRÁLY Sándor

Eszterházy Károly University, Faculty of Informatics, Institute of Mathematics and Informatics, Department of Information Technology, Eger, Hungary,
e mail: király.sandor@uni-eszterhazy.hu

About this document

Published in:

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

Volume 2, Number 2. 2020

ISSN: 2676-9425 (online)

DOI:

10.36427/CEJNTREP.2.2.833

License

Evaluation of Block-Based and Text-Based Coding

BERNÁT Péter

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

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

1. Block-based and text-based coding

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

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

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

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

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

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

2.1. Inserting a language element into the code

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

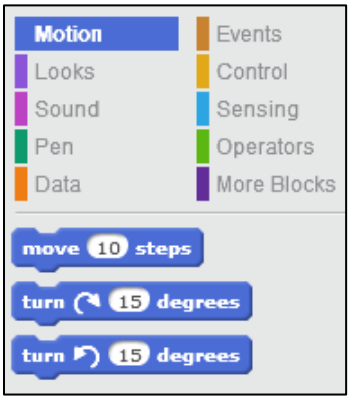


Figure 2: Categories of Scratch blocks, and some commands of the Motion group

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

Figure 3: Automatic code completion in Small Basic

2.2. Giving arguments to a language element

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

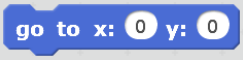
Scratch	Imagine Logo
	`stamp`
	`setpencolour`
	`setxy`
	`word`
	`and`

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

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

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

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

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

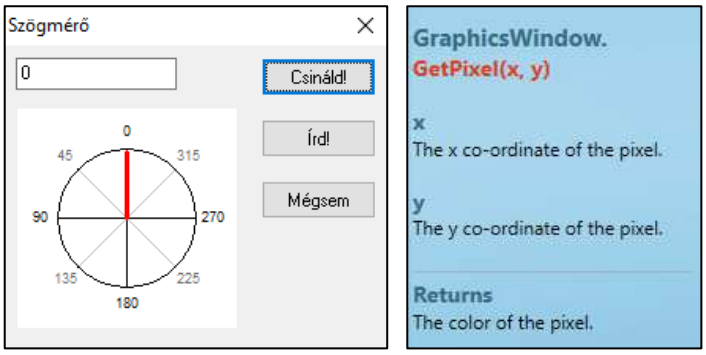


Figure 4: Form to help parameterize a command in Imagine Logo,
and context-sensitive help about parameterizing the command at the cursor in Small Basic

2.3. Creating a control structure

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

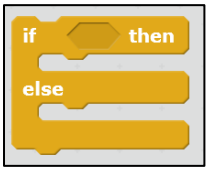


Figure 5: Selection in Scratch

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

2.4 Syntactic correctness and indication of syntax errors

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

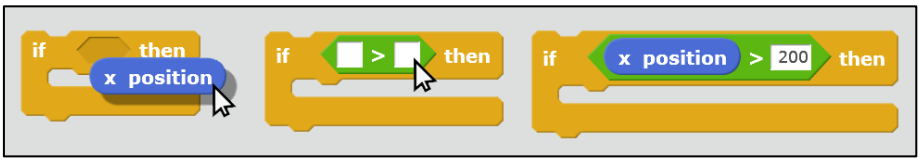


Figure 6: No numeric value, but a logical expression may be the condition of a selection in Scratch

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

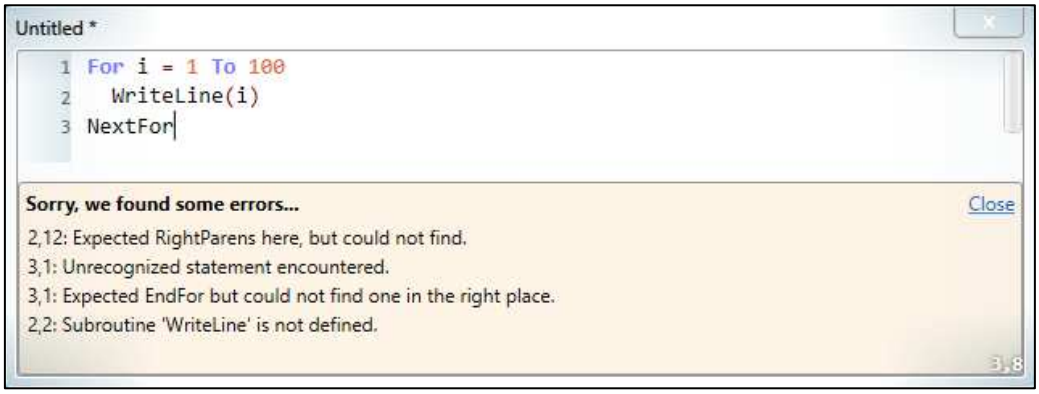
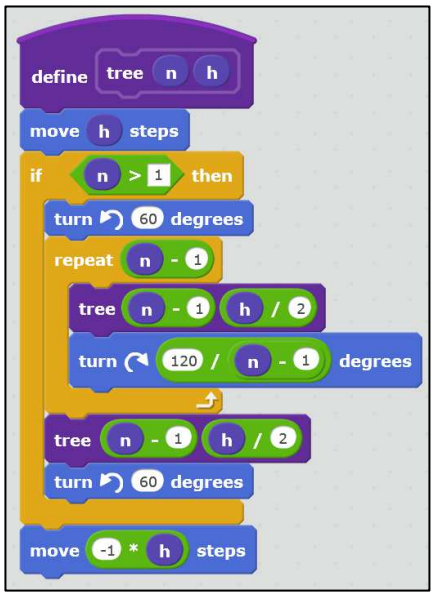


Figure 7: Error messages appear when starting the program in Small Basic

2.5 Readability

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




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

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

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

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

2.6 Speed of entering code

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

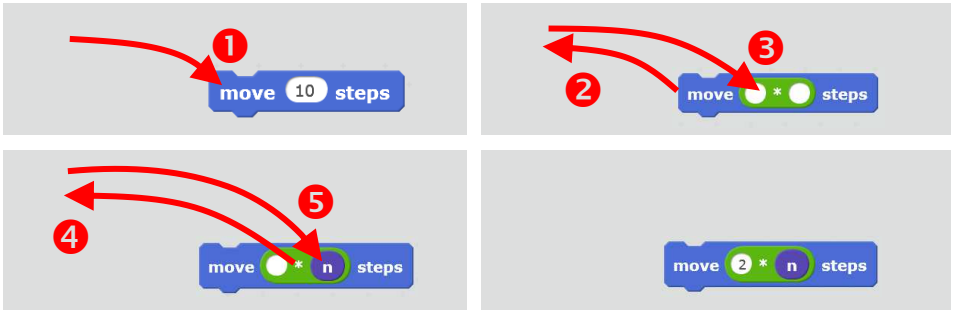


Figure 9: Mouse movements between the block set on the left side of the screen (not shown) and the programming area required to assemble the instruction shown in the lower right phase in Scratch

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

2.7 Extent of code

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

2.8 Editing in other environments

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

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

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

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

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

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

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

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

4.1 Arguments for block-based coding

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

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

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

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

4.2 Arguments for text-based coding

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

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

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

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

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

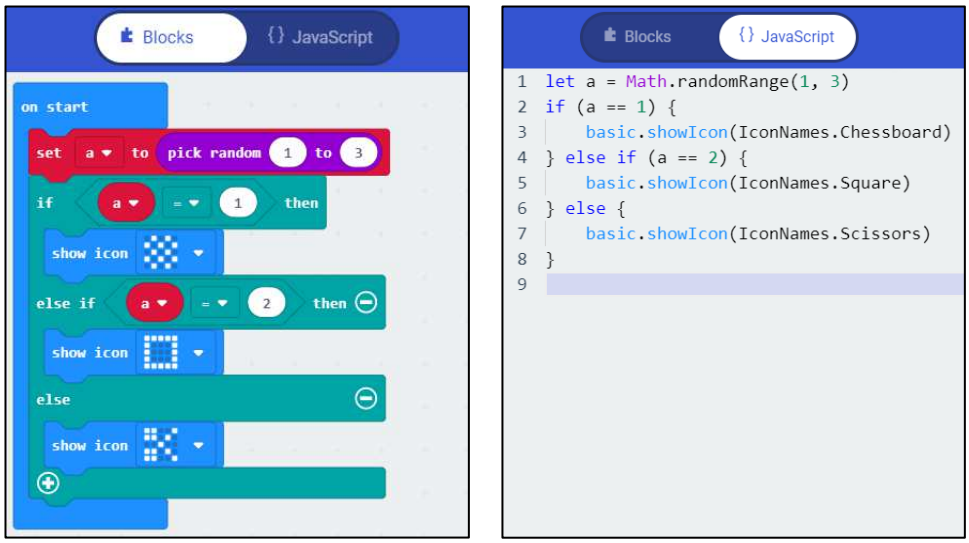


Figure 10: The same program in block and text view in Microsoft MakeCode

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

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

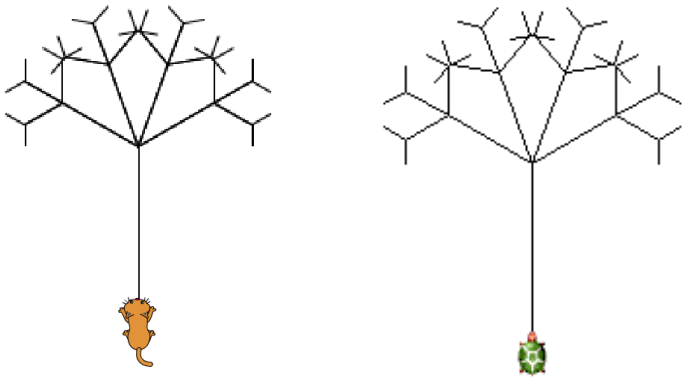


Figure 11: The result of the two matching programs shown in Figure 8 in Scratch and Imagine Logo

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

6. Interviewees' experiences on transition

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

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

7. Conclusion

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

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

Bibliography

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

Authors

BERNÁT Péter

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

About this document

Published in:

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

Volume 2, Number 2. 2020

ISSN: 2676-9425 (online)

DOI:

10.36427/CEJNTREP.2.2.471

License

Introducing the Gender Aspect into Hungarian IT Education[1]

SZLÁVI Anna

Abstract. The IT sector has been rapidly growing; in fact, a high number of jobs are expected to remain unfilled, which makes it our common interest to attract more students to this field. The striking scarcity of women in the IT sector offers an obvious solution: if more women could be involved, the labor shortage in IT could be fixed. The low number of women in IT has strong ties with IT education; therefore, the role and responsibility of teachers is immense. If we want to have more workforce and more competent professionals in the IT sector, educators need to create a more hospitable environment for girls and women in (both secondary and tertiary) education. The present study aims to show why it is key to introduce the gender aspect into IT education, both globally and specifically in Hungary. The primary goal of the article, thus, is to call attention to the repercussions of entrenched gender stereotypes in education, while also offering specific methods that can make IT-related classes more inclusive.

Keywords: IT, IT education, STEM education, gender, stereotypes

1. Introduction

The IT sector has been on the rise. In fact, it has been predicted to grow by 7.6% by the end of 2020, which means that almost 800,000 vacancies will remain unfilled in Europe. [1] At the same time, only around 16.5% of ICT professionals are women in the EU28 zone according to the statistics of Eurostat, the European Commission's primary database, as of October 2019. In Hungary, the situation is particularly severe: the EC's statistics reveal that in 2018 women constituted only 9% of those employed in the sector, which is the lowest in the EU28 zone. [2] With a better involvement of women in the tech industry, the EU's – and particularly Hungary's – GDP could be increased by millions of euros each year, EC concludes.

The present scarcity of women in IT has obvious ties with IT (and overall STEM) education. The stereotypes that link science to men, and thus, to boys, are revealed to be primary causes of women's, and girls', exclusion from fields connected to science. [3] According to Nosek et al.'s cross-national research, cultures that tie gender (specifically masculinity) to science discourage women to be accepted and perform well in STEM[2]. The role and responsibility of educators, therefore, is immense. If we want to have more workforce and more competent professionals in the IT sector, educators need to create a more hospitable environment for girls and women, both in (secondary and tertiary) education.

It has been declared that IT classes need to be diverse [4], interactive [5], and cooperative [6]. The present paper claims that this is not enough. IT classes, in addition, need to be sensitive to social identities and social stereotypes as well, including those connected to gender. It is vital that the educator considers not only the unique skills and competences of students, but also their cultural, ethnic, or gender identities, counterbalancing possibly harmful social stigmas. The present study, thus, aims to show why it is essential to introduce the gender aspect into IT education, both globally and specifically in Hungary. The primary goal of the article is to call attention to the repercussions of entrenched gender stereotypes in education. Revealing that the low representation of girls and women in IT is not only a theoretical problem but social and technological as well, the paper

[1] Throughout the paper, "IT education" will be used as an umbrella term that refers both to secondary Informatics education and tertiary Computer Science education.

[2] STEM is the acronym of Science, Technology, Education, and Mathematics.

pursues to offer specific methods and practices that can make IT-related classes more inclusive, thus involving more girls and women in the IT sector, specifically in Hungary.

2. Theoretical background

2.1. The concept of gender

In the past decades, numerous international and Hungarian researchers turned their attention to the issue of gender. [7,8,9,10] Various disciplines, such as linguistics [11], social sciences [12], psychology [13], and pedagogy [14], have revealed that gender inequality and discrimination is pervasive: globally women are at significant disadvantage, compared to men, in a number of areas of life. For example, girls are left out of the education system more often and/or earlier than boys due to child marriage, pregnancy, family duties, sanitation problems, or poverty, among others [14]. When growing up, women are less likely to work in well-paid positions, or when in the same position they are typically paid less than men. [15]

As for leadership and politics, women have a much lower representation worldwide than men (see [16]). Apart from education and empowerment, women are in a less advantageous position regarding safety as well: 1 in every 5 women is a victim of domestic violence, perpetrated by a male (see [17]). In addition, even after the assault happened, women get less than necessary protection: the discourse of trying rape and harassment cases still often involves victim-blaming. [18] In sum, women tend to lack equal opportunities regarding education, wealth, power, protection, and safety. It is so mainly because of mainstream ideology, based on binaries and gender stereotypes, which creates and maintains a gender-based hierarchy. [19]

As for the Hungarian context, the country has been part of the OECD since 1996 and the European Union since 2004; as such, Hungary is supposed to be among the most progressive communities in the world regarding human rights. If we look at the statistics, however, we are faced with a different reality. According to a recent worldwide survey [20], Hungary is among the 40 countries with the largest gender pay gap in the world, and among the 10 with the worst political representation for women. Both of these results place Hungary as one of the worst both within the OECD region and the EU concerning equal opportunities.

Patriarchal gender relations have a long history in Hungary [21]; thus, gender inequality is not a new phenomenon. Nevertheless, in the last decade gender equality became one of the most severe within the region. While in 2006 the country held the 55th position on the above mentioned Global Gender Gap list, 10 years later Hungary doubled its score, positioning itself in the worst third. In other words, it is crucial to address gender inequality in Hungary, and particularly women's low representation in leadership and power, in order to be able to influence change. In her recent comprehensive study, Szlávi analyzed contemporary Hungarian discourses to reveal the depths of entrenched gender stereotypes, concluding that all main domains of popular communication, including language use, media, advertising, and even art are imbued with gender-based inequalities. [22]

2.2. Gender in IT

Both globally and locally, gender inequalities pose serious problems to our society. The IT sector, one of the economically most influential industries in the world, is no exception. Women are at a disadvantage even in this field. If we observe the highest level of the sector, we will find a low

number of women in the management positions of the world's three most prominent IT companies; Apple, Microsoft, and Amazon. At Apple there are 4 women for 12 men, at Microsoft 3 women for 12 men, and at Amazon 6 women for 11 men (see apple.com, microsoft.com, aboutamazon.com).

Nevertheless, it is not only the management level that reflects severe inequalities, but it is the whole of the IT world. Kirkup's survey, studying the IT sector of the UK, the USA, Canada, Taiwan, Ireland, and Spain, uncovered both horizontal and vertical division of labor and inequalities within the sector. [15] The main finding of the research is that women are underrepresented in IT professions. For one, there are emphatically fewer women in the technical or engineering segments of the field; instead, they typically work in non-professional roles such as sales (horizontal inequality). For two, the women that do get employment in the IT sector are typically paid less and/or work in a lower status than their male colleagues (vertical inequality). For three, in certain countries it also appeared to be a trend that women that end up working in the sector have a higher degree than men. Finally, Kirkup's research revealed that not only employment but also advancement shows different patterns when it comes to gender: men are promoted faster than women even if their leadership skills are reported to be weak.

More and more scholars [23,24,25] are turning their attention to the low representation of women in the field, which keeps raising awareness. Thanks to this, we can witness an increasing number of initiatives that attempt to turn the findings into a call for action, with the goal of enhancing the participation of girls and women in STEM fields. Besides the large-scale global projects[3], we can list some initiatives that strive to ameliorate the acute inequality in these fields even in Hungary. As an example, see "Lányok napja", that is, 'Girls' Day' (http://lanyoknapja.hu). "Lányok napja" is an annual nation-wide event dedicated to promoting the STEM sector for high school girls through diverse programs offered to them by universities and companies, that is, their potential places of education and employment.[4]

Overall, though, the attempts to involve a fair amount of girls in the field are limited in success. There are still just a handful of women in – well paid and / or influential – IT positions, which leads to a shortage of role models for the younger generations. Role models play a crucial role in inspiring students to pursue a certain field. When examining the impact of role models in the media about science, specifically mathematics, Lócska and Kovács concluded that the attitudes of high school students towards maths was positively affected by popular movies such as *Hidden Figures* [26]. The 2016 American film is (mainly) about the contributions of three African American women to NASA's aeronautical success. The movie not only portrays the STEM field in an attractive light but it also pays tribute to the role of women in the field. While such works of popular culture are undeniably inspirational, the scarcity of visible women role models in present-day IT poses a serious obstacle. Without approachable and living female role models, girls are left without the encouraging evidence that it is a woman's path too, thus few of them venture into the field. In addition, the discouraging or even hostile attitudes, of some men and boys, towards women that do get into IT remain unchallenged, making it hard for women to remain in the sector. In other words, the gender stereotypes that brought about, and keep on maintaining, the unequal situation, namely that women are not good at objectivity and logic, hence science, nor are they interested in such fields, remain unquestioned. [15]

[3] Some such projects are girlsintech.org, girlswhocode.com, and www.womenintechnology.org among others.

[4] Originally, Girls' Day is an European initiative dating back to 2000, which Hunary joined in 2012.

It must be noted that the absence of women in IT is not just a matter of social injustice. Apart from the fact that the low representation of women in this well paid and highly prestigious profession makes the overall and pervasive gender gap even more serious, it also has tangible and concrete technical repercussions. Who makes the software, the program, and the algorithm influences the software, program, and algorithm itself. If women are typically only users, and not developers, of IT [27], the female perspective, or more broadly diversity, will be missing from the process of creation and development.

The lack (or scarcity) of women, or generally that of diversity, in the programming team poses problems in several domains of IT. To demonstrate this, let us observe a popular field of AI technology, that of facial recognition. Facial recognition, which uses machine learning, works properly if it operates in an accurate and unbiased way, which entails gender-neutrality as well. Yet, according to recent research done by MIT, most face recognition softwares (Microsoft, Face++, and IBM) fail to operate in an accurate, unbiased, and gender-neutral way. Buolamwini and Gebru pointed out that these softwares do not work with the same precision and efficiency when it comes to the faces of women, especially women of color, as they do with (white) men [28]. Their error rates of the above listed facial recognition systems were 23%, 36% and 33% (respectively) when recognizing black women's faces, while with white men there was hardly any error.

According to Buolamwini and Gebru, the training sets used in machine learning were not adequate: (gender and ethnic) diversity was underrepresented in the samples, and likely in the programming teams as well [28]. Buolamwini, furthermore, warns that stereotypes and biases – or invisibility that maintains lower social status – can easily get into the algorithms, thus into the programs as well, further promoting and reinforcing injustice. It is especially the case if the programming team is not diverse enough so the members could watch out for the blind spots, or the unconscious biases, of each other. [29]

Facial recognition systems are pervasive, as they can be used for a myriad of purposes. To mention their most widespread usage, it is worth noting that even smartphones are equipped with general face recognition software nowadays. But the same face recognition technology forms the basis of more specific applications as well; for example, emotion recognition programs, enabling people on the autism spectrum or people with visual disabilities, are built on this technology as well. In addition, crime prevention and policing also applies similarly trained facial recognition programs [28]. It is easy to see that if the basic face recognition algorithm is biased (regarding gender, ethnicity, or anything), it can have severe consequences in such situations; misidentifying a perpetrator is not a minor mistake, sometimes not even one that can be undone.

As the study about facial recognition shows, a diverse, inclusive, and cooperative developer team is undeniably vital. It is for a broader social good that gender-inclusivity and diversity must be put as a priority. In addition, business-focused research seems to suggest the same, only on different grounds. Frehill and McGrath Cohoon's investigation points out that diverse teams perform better and bring innovation [23]. Thus, they reiterate that it is not only for a social good but also for the economy that diversity should be addressed and enabled.

2.3. Gender in IT Education

It is clear that the involvement of women, just as other social groups, in IT is our common – social, economic, and technological – interest. To enhance gender balance in the field, IT education has a central role.

According to Ramirez and Kwak's analysis of UNESCO's statistics about university enrollments between 1970 and 2010, women's participation in tertiary programs has been steadily rising all over the world [30]. In 2010, women comprised already 50-60% of all enrolled students. As for STEM programs, however, the improvement was less significant: while there is some global progress regarding women's involvement, it is not as much as what we can see regarding women's overall presence at universities. In 2010 women still failed to constitute half of the students pursuing STEM majors; the numbers were stuck at 30-40%. In Hungary, women's participation has always been lower; around 24%. What is more, Hungary is one of the four countries, on the list of more than a hundred, where the participation of women in engineering and science majors not only failed to make a leap but it actually decreased during the past decades: from 24.7 to 23.1%.

In 2012 a Hungarian technical university decided to conduct a large-scale survey in order to find out why Hungarian high school girls are reluctant to choose engineering majors [31]. As the main finding of the research it was concluded that girls decide to stay away from engineering/IT mainly because they had bad experiences in high school STEM classes. Many of the high school girls who were asked complained not only about the content of physics and maths (and other STEM) classes but also about the attitudes of their teachers. Some of the educators involved in the survey confirmed these negative stereotypes themselves by making generalizing statements such as girls are worse at science, girls are less smart, etc. In fact, a number of these teachers were aware of their tendency to treat students differently based on their gender.

The research was interested in uncovering girls' attitude towards IT as well. Some of the main findings include that girls labeled IT and engineering as "masculine" fields. They also added that in their opinion reconciling such careers with "feminine duties" such as family life was hard.

From the above statements it is apparent that gender roles as social constructions [8] have a great impact on the career choices of students. In other words, one of the key findings of the study was the connection of gender stereotypes with the low number of women in STEM.[5] Whether educators feed or challenge such stereotypes largely determines what professions students can imagine suitable or appropriate for themselves. Therefore, if IT teachers strived to create gender-neutral environments, or ones which openly defy social stereotypes, it could positively affect the career choices of their female students, while also sensitizing male students.

3. Gender-Inclusive Practices in IT Education: A Case Study

3.1. Background

After identifying the problem, it is crucial to discover what we can do to mend gender imbalance in IT education. According to Bonder, there have been three different approaches to deal with the issue, which are: "to fix the numbers," "to fix the institutions," or "to fix the knowledge." [32]

The first approach entails that we focus on the very result; that is, after acknowledging the problem that there are few women in IT programs, we attempt to simply increase their presence in the student body. Even if achieved, this, however, does not guarantee that the change will be sustainable. Mainly so because the infrastructures of the institutions will remain unchanged; that is, still more adapted to men and more challenging for women (to name some of such elements: the

[5] To read a more detailed analysis of the research, check [33].

overwhelming majority of male instructors, the advancement inequalities, or the lack of facilities that would enable women with young children to complete a program).

Realizing that only some of women's challenges can be mended simply by adding more female students into the institutions, the second approach, "to fix the institutions," turns to overcoming the institutional barriers that keep women away from the majors and the field. Thus, this approach focuses not only on the result but also on some of the reasons that brought about those results (for example, by creating nursing rooms for women with children). Unfortunately, however, even this approach fails to address the roots of the problem, the social conception of structured gender inequality, which jeopardizes a long term success.

The third approach, which proposes the concept of "fixing the knowledge" in order to mend women's scarcity in IT, tries to go to the root of the problem. According to this idea, only if we educate people about gender balance starting as early as in school can we bring about sustainable change. How an educator can make inclusive IT classes is not an easy question, though. In the following, we will go through some of the main theoretical concepts, then a case study will demonstrate one of the possible ways they can be put into practice.

It is emphatically important how educators in primary and secondary schools approach their classes, as students are still in the phase when they can decide for or against pursuing an IT career. For a primary or secondary school teacher, therefore, it is crucial to be aware of the gender stereotypes related to their field, along with their potential impact. To do that, they first of all need self-reflection, as Lipovits et al. point out [34]. In addition, tolerance, open communication, equity, fairness, and social sensitivity [34] are also crucial, not only for enhancing gender balance but also to create versatile, diverse, and cooperative working groups.

Next to the educator's attitude, teaching methods have a central role as well. A diverse student group can be motivated and inspired by diverse tasks [15]. In other words, presenting from multiple angles what IT is capable of is more likely to engage students and reveal the true potential of technology (and a career in technology). It is also necessary to realize that solving tasks in pairs or in groups, that is, in cooperation, makes the learning process more effective [6] and more motivating. Additionally, the content of the tasks should also be chosen consciously: multicultural study materials can be inspiring and at times even liberating. Kátai suggests the use of training materials with ethnic and cultural content [35]; in a similar vein gender diversity could also be an important element of our materials.

In summary, it is an essential goal, and actually a requirement, of 21st century educators of primary and secondary schools to help girls become creators, not only users, of technology. It should be a conscious aim to present not only why IT is exciting and useful, but also that it is an available career path for girls (too). As a consequence, raising students' awareness of role models and success stories related to women in IT can make the field more attractive and more widely chosen.

University educators have much fewer students to help thrive in or deter from IT careers, yet their role and responsibility cannot be underestimated. Their task is not primarily to raise interest but more to keep students in the field and to educate them to become conscious, socially sensitive, and inclusive IT professionals. In order to do this, Margolis and Fisher claim that it is essential to humanize and contextualize programming [36]. What this means is that university educators need to demonstrate the social aspects of IT, on the one hand, and to connect it with various fields and disciplines, on the other. Next to these principles, the authors consider teamwork and cooperation to be the methodological pillars of inclusive tertiary IT education. It is so because such settings can encourage tolerance and inclusion to be at the core of the programming process. Finally, the authors underscore the importance of providing female students with role models, which can be

done through mentorship programs in which students are teamed up with female scholars or professionals. [36]

3.2. Case Study

The following case study from the author's own teaching practice is to demonstrate how an IT-related course can be put together using methods that support gender-inclusivity and diversity. First the setting of the course will be explained, then the material and methodology will be detailed.

In 2018 a new, four-semester-long Computer Science (CS) program was launched at the Faculty of Informatics at Eötvös Loránd University, Budapest. In the frames of the program, students are expected to learn and practice professional English language use, not just programming, given that most of the CS literature is in English. The two-semester-long language course, scheduled for the first two terms of the program, is compulsory; that is, all students take part in it and, what is more, at a crucial time of their studies. Depending on the current enrollment, the groups of the course are made up of 12-18 students; in 2018/2019 that meant 2 groups, while in 2019/2020 there were 4 of them. The present case study is intended to introduce this course, running for two years now.

The course, whose purpose is to acquaint students with CS topics and materials in English from around the world and make them engage in discussions related to IT, is not an IT class strictly speaking, at least not one about hard skills. Despite this, the principles and practices that will be presented are useful and usable in hard skills, programming courses as well. For one, because the course is in fact very tightly connected to IT, as its primary goal is to bring IT topics closer to and more accessible for CS students early in their studies. For two, because the principles and the methods that underlie this course are general, thus transferrable to other types of classes.

For designing the course, Margolis and Fisher's recommendations [36] served as methodological tools. In order for IT classes to be efficient, the authors underline the importance of using teamwork and cooperation when solving tasks; humanizing and contextualizing programming; and presenting role models as part of the study material. The specific circumstances of the course also had to be taken into consideration. There were two factors that emphatically called for the application of Margolis and Fisher's tools: on the one hand, the large number of students, with different competence levels, within a group made it impossible to plan a lot of – undifferentiated – frontal classwork. Instead, group and pair work seemed to be more suitable as that way students could and were forced to help and complement each other while solving tasks together. On the other hand, as the student body was (and has been) overwhelmingly homogeneous regarding gender and ethnicity (the majority has been white, male, and Hungarian), it was also obvious that diversity, difference, and inclusion had to be brought into the class through the study materials. Altogether, both with the methods (of differentiated education and teamwork) and the contents (of diverse materials and contextualized tasks), the course was aimed to improve students' cooperational and social skills, in order to make them better language users, but also better programmers.

As for the content of the course, it was designed primarily to demonstrate the human side, the social context, and the diversity of IT to students who are interested but may not yet be deeply experienced in IT. The course was divided into two parts (semesters) in a way that the materials did not only connect to but also derived from each other. While the first part was supposed to build the foundations of the coursework, the second was meant to elaborate on and deepen the concepts and the skills. All the communication, skills development, or analytic activities of the

course were based on eight, high-quality, English-language TED-talks[6] covering a specific IT topic. They are the following:

- Agüera y Arcas, Blaise: How PhotoSynth can connect the world's images
- Arar, Raphael: How we can teach computers to make sense of our emotions
- Bracy, Catherine: Why good hackers make good citizens
- Buolamwini, Joy: How I'm fighting bias in algorithms
- Feinberg, Danielle: The magic ingredient that brings Pixar movies to life
- el Kaliouby, Rana: This app knows how you feel – from the look on your face
- Lupi, Giorgia: How we can find ourselves in data
- Redmon, Joseph: How computers learn to recognize objects instantly

In the middle of the coursework and at the very end of it, students were asked to do self-reflection related to the course material. Both in graded and ungraded frameworks, they were asked to share their opinions on ideas such as "cooperation," "social responsibility," "tech values," and "role models" in the connection of the analyzed tech talks, and generally IT.

3.3. Discussion

The course material was compiled in order to reflect and emphasize the diversity, the rich variety of contexts, and the human aspects of IT, so students can more easily find themselves in and connect with CS.

As one of the guiding concepts of the course, diversity appeared in the material on several levels. First of all, the speakers were selected so they can represent and present different subtopics within IT in order to showcase the wide options within the field. Next to expertise, the identity of the speakers were also meant to be a testament of difference (and inclusion). Regarding their ethnicity and origins, the speakers are not at all homogenous: there are white and black tech experts; European, American, and African IT professionals. Religion and sexual orientation appear as a matter of choice as well. Finally, as far as gender is concerned, it was a very conscious decision to include a fair number of women among the presenters. Five of the eight speakers are women (C. Bracy, J. Buolamwini, D. Feinberg, R. el Kaliouby, and G. Lupi), as figure 1 shows. With this, the course aimed to clearly demonstrate, both to the few female students and to the masses of male students, that women also have a place in IT. By watching such role models speak, students could realize that women can be authentic, inspirational leaders and experts of IT as well. Altogether, the course material was designed to make (both male, female, and non-binary) students conscious of the fact that IT is, and has to be, a diverse field, in the gender aspect and many others.

[6] TED (www.ted.com) is a series of conferences mainly about Technology, Education, and Design (hence the acronym), whose high-quality and mainly English-language talks are freely accessible on their website. For most of the talks, subtitles and transcripts are also available, which makes them not only thought-provoking professional presentations but also easy to use language learning materials.

Figure 1. The female presenters of the IT-related TED talks used in the course

As Margolis and Fisher emphasize, it is also essential to contextualize IT [36]. As it was mentioned before, all the talks present a specific and different area within technology, such as computer vision, object recognition, animation, data visualization, and so on. Each of them place their respective technology within a specific context, offering insight into what can be done with coding. To give an example, one of the speakers (el Kaliouby) demonstrated in her talk that machine learning can be used for the development of emotion-enabled apps which can facilitate emotion recognition.

Besides giving context to IT, it was also an important goal to humanize it. Even in this aspect all the selected talks fare well: each of the speakers stress the importance of social responsibility when putting their specific technology to practice. To demonstrate this, it is best to continue elaborating on the previously mentioned video. When talking about her company that works with emotion recognition, el Kaliouby was quick to emphasize the social relevance of such technology. She explained that an emotion enabling app can not only help people who are visually impaired or are on the autism spectrum decode emotions, it can also assist all of us to communicate and connect better with our tools and each other. The fact that every speaker approached their topic in a similar matter, not failing to mention the social importance of their projects, students could become aware that social responsibility is key in IT, even if this aspect is rarely addressed during their CS studies.

In order to check the effectiveness of the methodology, students were asked to reflect on various concepts, core to the course, multiple times. Both at the end of the first term and at the end of the second term, in practice and in test situations they had to contemplate on (1) what they think about cooperation and teamwork, (2) what they consider to be the main social values in IT, and (3) who they were inspired by. This way, the course not only offers them situations to practice the values Margolis and Fisher define to be effective learning tools, such as groupwork, the humanization and contextualization of IT, and role models, but it also makes them become conscious of them.

A selection of students' short self-reflective essays is found in the Appendix. Three of the main topics are listed: in Appendix 1 their opinion about collaboration, in general and in IT in particular, is summarized; in Appendix 2 their stance on social values and tech values is detailed; and Appendix 3 lists their confessions about their role models as future IT experts.[7] Going through the entries in Appendix 1, we can see that students believe that cooperation is crucial not only for efficiency and swiftness, but also because team members can complement each other and produce more diverse

[7] The excerpts were not categorized according to the students' gender, for one, because sensitivity about the gender-inclusiveness of IT is desired from both (or all) genders; for two, because students were not asked to specify their gender during the course. The selection aimed to highlight the diversity of their responses, thus avoiding the binary bias of gender stereotypes.

and socially more inclusive solutions. As for values in IT, students listed equality, inclusiveness, diversity, and lack of discrimination. Some of them even addressed that it is their responsibility as future programmers to respect and enforce these values. Finally, when students were asked which speaker inspired them throughout the course, they responded in a lengthy and passionate way. It is also worth noting that several speakers were named, not just one, which shows that the diversity of people can be inspired by the diversity in the course material (Appendix 3). It is worth noting that multiple students underlined that they could relate to the personal stories (often about initial struggles) of the speakers and that seeing them succeed gave them hope that they would too. Especially female students seemed to verbalize such impressions.

In sum, with the course I have made a conscious attempt to place IT in a setting that embraces diversity, overcomes stereotypes, and celebrates social responsibility, both with its choice of content and methodology. Besides enforcing the general idea of diversity, the course specifically strived to provide support and inspiration for female students, through the presentation of female role models and the discussion of equity and equality in IT, so that they persist and thrive in the IT field.

4. Conclusions

The goal of the present paper was to explain why it is important to approach IT education from a gender perspective. Due to social stereotypes infiltrating classrooms (as well), girls and women are gravely underrepresented in the IT sector, which has serious social, economic, and technological ramifications. The role of IT educators is huge in this process. By questioning entrenched gender stereotypes related to IT (and STEM in general), along with enabling diversity and cooperation within the classroom, teachers can have a beneficial impact on the gender balance in the IT world. The second half of the paper aimed to give some practical guidance about how this can be done. The case study of a recent, and as a matter of fact ongoing, IT(-related) course of the author's was meant to show how the notions of cooperation in classwork, the contextualization and humanization of the IT topics, and the presentation of role models can be implemented to overcome gender stereotypes and celebrate diversity.

Acknowledgements

The author wishes to say thanks to Zsuzsa Kecskés, Amy Soto, Péter Szlávi, and Ráhel Turai for their invaluable contribution to the paper.

Appendices[8]

Appendix 1: Collaboration

1) "I think collaboration in general is important because people have different abilities and talent. If they work together, anyone can add something. By this they can create a better result. On the other hand, it's important because teammembers came in different environment. They have diverse thoughts, feelings, ideas. And because of that, they can reduce prejudice and bias."

[8] The essays were not corrected or modified for the sake of authenticity.

2) "We need to unite as people to accomplish far greater things than before. It would be faster, more efficent. I personally feel more happy if I could see bigger and better collaborations with time."

3) "I think collaboration is important because we all have different ideas and opinions. By working together we can assure that the final product of our work will reflect us all. Collaboration is specially inportant in programming, because usually a person only works on a specific part of the project, so we should be able to work together in order to finalize the project successfully. Based on the Civic Hackers video, and also based on real life experiences, when people work together for one common goal, they are more successful than as individals. If a lot of people join togheter, they can make real changes in the world."

4) "In my oppinion, collaboration is the most important thing in life. Also, if you collaborate with others, you can get other ideas, other visions about a thing. In Civic Hackers, they involved everyone. In class, we devided the task and everyone worked alone on a part, and then we discussed the whole thing. I think this is one is a better way of collaboration."

Appendix 2: Values

1) "There are some issues, that we have to fight against: neutrality, privacy protections, and internet acces (for not everyone)."

2) "Some tech values would be inclusiveness and diversity, so the technology can be accessed and used by the widest possible audiences, without discrimination."

3) "As we get more and more dependent on digital technology, it is clear, that programmers and software developers have way more power, but also more responsibilities. We/they have to recognize this, otherwise they can contribute to bad practices such as racial discrimination as we seen in the algorithmic bias video. However, the same power can also be used for benefiting society (as we seen in the civic hackers video). Developers have to have values in mind when innovating, to keep technology easily accessible and also beneficial for all people."

4) "Tech must serve everyone in an equal way, nobody should be discriminated by tech. Tech should focus on people and it must connect them together not separate them. Tech is should be more personal and it don't have to be cold."

Appendix 3: Inspiration

1) "I found Danielle Feinberg and her talk about her work at Pixar very interesting, and inspiring. How she struggled with her future plans, and found her way to the work she does now was familiar to me. And the way she described how the computer science and programming knowledge she learned could be used to express her artistic desire and passion was truly heartwarming and inspiring."

2) "One of my inspirations was the girl who talked about algorythmic bias. Before I saw that video I havent thought about this problem before, but after viewing that video, I just thought that this should get more attention."

3) "I liked the presenter who talked us about civic hacking the most, because she showed us a new way to participate in activities, which can help for every member of the society, and many small acts can solve huge problems of the government."

4) "Personally, I found the Pixar movie video the most inspiring. I can relate to the talker being interested in art and programming simultaneously, and her desire in wanting to conbine them. At the moment I am learning animation and film making, and I also have a sertificate in photography. By learning programming I can widen my future possibilities at pursuing a carrier. It can help me create my own website for my portfolio. And maybe later I will learn to make computer games by combining my artistic and programming skills. To see someone who have already done that and has been successfull, gives me hope for the future."

5) "The one who had the biggest effect on me was clearly Catherine Bracy. She showed how action of the everyday people can make a big difference. She presented many interesting examples, where people stood up for themselves, made better systems for their own good and contributed to public services. Sometimes it just seems like people are not willing to sacrifice the time and energy to make change, they only complain, and wait for the change to come form above. However, she showed us that if the platform is set up (like the Oklahoma "write-a-thon" or the challenge issued in Mexico city) people are happy to contribute. As a future programmer, this infected me with new ideas about how I could make such "platforms" in the future, and how to get people involved."

6) "I was inspired the most by the Pixar women. She and her childhood dream gave me a push that I needed in programming. I love to draw too, just like programming, and according to her, some day maybe I can bring these two togeather. My biggest dream is to become a Web designer or a Game developer, and if she could make that, I can do it too."

Teaching materials

Agüera y Arcas, B. (2005). How PhotoSynth can connect the world's images. *TED*. https://www.ted.com/talks/blaise_aguera_y_arcas_demos_photosynth
Arar, R. (2017). How we can teach computers to make sense of our emotions. *TED*. https://www.ted.com/talks/raphael_arar_how_we_can_teach_computers_to_make_sense_of_our_emotions
Bracy, C. (2013). Why good hackers make good citizens. *TED*. https://www.ted.com/talks/catherine_bracy_why_good_hackers_make_good_citizens
Buolamwini, J. (2016). How I'm fighting bias in algorithms. *TED*. https://www.ted.com/talks/joy_buolamwini_how_i_m_fighting_bias_in_algorithms
Feinberg, D. (2015). The magic ingredient that brings Pixar movies to life. *TED*. https://www.ted.com/talks/danielle_feinberg_the_magic_ingredient_that_brings_pixar_movies_to_life
el Kaliouby, R. (2015). This app knows how you feel – from the look on your face. *TED*. https://www.ted.com/talks/rana_el_kaliouby_this_app_knows_how_you_feel_from_the_look_on_your_face
Lupi, G. (2017). How we can find ourselves in data. *TED*. https://www.ted.com/talks/giorgia_lupi_how_we_can_find_ourselves_in_data
Redmon, J. (2017). How computers learn to recognize objects instantly. *TED*. https://www.ted.com/talks/joseph_redmon_how_a_computer_learns_to_recognize_objects_instantly

Bibliography

1. Hüsing, T., Korte, W. B. & Dashjae E. *e-Skills in EuropeTrends and Forecasts for the European ICT Professional and Digital Leadership Labour Markets* (2015-2020). Empirica. http://eskills-lead.eu/fileadmin/lead/brochure-lead/working_paper_-_supply_demand_forecast_2015_a.pdf(accessed: 03/20/2020)
2. Eurostat. *ICT specialists in employment.* (2019) https://ec.europa.eu/eurostat/statistics-explained/index.php/ICT_specialists_in_employment#ICT_specialists_by_sex (accessed: 03/20/2020)
3. Nosek, B.A., Smyth F. L., Sriram, N., Lindner, N. M., Devos, T., Ayala, A., Bar-Anan, Y., Bergh, R., Cai, H., Gonsalkorale, K., Kesebir, S., Maliszewski, N., Neto, F., Olli, E., Park, J., Schnabel, K., Shiomura, K., Tulbure, B. T., Wiers, R. W., Greenwald, A. G. *National differences in gender-science stereotypes predict national sex differences in science and math achievement.* PNAS June 30, 2009 106 (26) 10593-10597 (2009). DOI: 10.1073/pnas.0809921106
4. Gál, B. & Dávid, Á. *Az önfejlesztés és a reflektív pedagógusszemélyiség kialakításának lehetőségei az informatika szakos tanárképzésben* (in Hungarian). InfoDidact, (2008). https://people.inf.elte.hu/szlavi/InfoDidact08/Manuscripts/GBDA.pdf
5. Pšenáková, I., Heizlerné, B. V., & Illés, Z. *Interaktivitás az informatikatanításában* (in Hungarian). InfoDidact, (2018). https://people.inf.elte.hu/szlavi/InfoDidact18/Manuscripts/PIHBVIZ.pdf
6. Nahalka, I. *Az általános didaktika és az informatikatanítás didaktikájának egymásra hatása* (in Hungarian). InfoDidact, (2009). https://people.inf.elte.hu/szlavi/InfoDidact09/Manuscripts/NI.pdf
7. de Beauvoir, S. *A második nem* (in Hungarian). Gondolat, (1969).
8. Butler, J. *Gender trouble.* Routledge, (2007).
9. Bozzi, V. & Czene, G. *Elsikkasztott feminizmus* (in Hungarian). Osiris, (2006).
10. Huszár, Á. *A nő terei* (in Hungarian). L'Harmattan, (2011).
11. Huszár, Á. *Bevezetés a gender-nyelvészetbe* (in Hungarian). Tinta, (2009).
12. Milestone, K. & Meyer, A. *Gender and popular culture.* Polity, (2012).
13. Kovács, M. (Ed.). *Társadalmi nemek. Elméleti megközelítések és kutatási eredmények* (in Hungarian). Eötvös kiadó, (2017).
14. Sperling, G. B. & Winthrop, R. *What Works in Girls' Education: Evidence for the World's Best Investment.* Brooking Institution, (2016).
15. Kirkup, G. *ICT as a tool for enhancing women's education opportunities, and new educational and professional opportunities for women in new technologies.* United Nations Division for the Advancement of Women (UNDAW), (2002)
16. OECD. *Women in politics (indicator)*, (2020) DOI: 10.1787/edc3ff4f-en
17. European Union Agency for Fundamental Rights (FRA). *Violence against women: An EU-wide survey*, (2014). https://fra.europa.eu/sites/default/files/fra_uploads/fra-2014-vaw-survey-main-results-apr14_en.pdf (accessed: 03/30/2020)

18. Ehrich, S. A megerőszakolt nem. A nyelv a szexuális erőszakról szóló tárgyalásokon (in Hungarian). In Juhász, V. &. Kegyesné Szekeres, E. (Eds.), Társadalmi nyelv és nyelvhasználat. Válogatott szemelvények az angol és a német szakirodalomból. (pp. 172-194): Szegedi Egyetemi Kiadó, (2011).

19. Bourdieu, P. *Masculine domination*. Polity, (2001).

20. *Global Gender Gap Report.* (2017) http://reports.weforum.org/global-gender-gap-report-2017/dataexplorer/#economy=HUN (accessed: 12/20/2019)

21. Marinovich, S., & Arpad, S. *Why hasn't there been a strong women's movement in Hungary?*, (1995) DOI: 10.1111/j.0022-3840.1995.2902_77.x

22. Szlávi, A. *The Construction of Gender in Hungarian Discourses.* Doctoral dissertation. Eötvös Loránd University (2019).

23. Frehill, L. M. & McGrath Cohoon, J. *Gender and Computing.* In: Pearson, Jr. W., Frehill, L. M. & McNeely, C. L. (Eds.) Advancing Women in Science. An International Perspective (pp. 237-264). Springer, (2015).

24. Mansour, N. & Wegerif, R. *Science Education for Diversity: Theory and Practice.* Springer, (2013).

25. Rosser, S. V. *Academic Women in STEM Faculty*. Palgrave Macmillan, (2017).

26. Lócska, O. D. & Kovács, Z. *14 to 18-year-old Hungarian high-school students' view of mathematicians appearing in the media - A case study*. In: Teaching Mathematics & Computer Science, 2018/2. pp. 183-194, (2018) DOI: 10.5485/TMCS.2018.0446

27. Crutzen, C. (2005). *Questioning Gender in E-learning and its Relation to Computer Science. Space for design, working, and learning.* In: Braidotti, R. & van Baren, A. (eds.) The Making of European Women's Studies Vol. VI. University of Utrecht, pp.40-59.

28. Buolamwini, J., & Gebru, T. *Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification*. In: Proceedings of Machine Learning Research 81:1–15 (2018). http://proceedings.mlr.press/v81/buolamwini18a/buolamwini18a.pdf

29. *Algorithmic Justice League* https://www.ajlunited.org/ (accessed: 12/20/2019)

30. Ramirez, F. O. & Kwak, N. *Women's Enrollments in STEM in Higher Education: Cross-National Trends, 1970–2010*. In: Pearson, Jr. W., Frehill, L. M. & McNeely, C. L. (Eds.) Advancing Women in Science. An International Perspective, pp. 9-26. Springer, (2015).

31. Papp, G. & Keszi, R. *A műszaki felsőoktatás a nemek tükrében – különbségek a pályaválasztás és az egyetemi tapasztalatok területén. Zárótanulmány*. In: Szekeres V. & Krolify Intézet (eds.) „Ti ezt tényleg komolyan gondoltátok?" Nők és a műszaki felsőoktatás (in Hungarian). Óbudai Egyetem, pp. 214–314, (2013)

32. Bonder, G. *Foreword.* In: Pearson, Jr. W., Frehill, L. M. & McNeely, C. L. (Eds.) Advancing Women in Science. An International Perspective pp. v-viii. Springer, (2015).

33. Nagy, B. *Háttérben: Kísérlet egy szervezeti nemi rend feltárására* (in Hungarian). L'Harmattan, (2014).

34. Lipovits, Á., Háli, A., Kovács, E., Pozsgai, T., & Gál, B. *Ötletek az informatikatanárok képzéséhez* (in Hungarian). InfoDidact, (2010). https://people.inf.elte.hu/szlavi/InfoDidact10/Manuscripts/LA_et_al.pdf

35. Kátai, Z. *Multicultural Computer Science Education*. InfoDidact (2011). https://people.inf.elte.hu/szlavi/InfoDidact11/Manuscripts/KZ.pdf

36. Margolis, J. & Fisher, A. *Unlocking the Clubhouse. Women in Computing*. Cambridge Mass, MIT Press, (2002).

Authors

SZLÁVI Anna

Milestone Institute, Budapest, Hungary,
e-mail: dr.szlavi@gmail.com

About this document

Published in:

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

Volume 2, Number 2. 2020

ISSN: 2676-9425 (online)

DOI:

10.36427/CEJNTREP.2.2.472

License

Classical Programming Topics with Functional Programming

VISNOVITZ Márton

Abstract. In traditional Hungarian programming education programming theorems are the foundation of learning programming. These generic algorithm patterns are traditionally introduced using the imperative programming paradigm with sequences, loops, and conditions. After learning about programming theorems and basic algorithms the next step is usually to go in the direction of object-oriented programming. One possible way to approach object-oriented programming is through enumerators: the implementation of enumerable data types and the application of programming theorems on them. To prove that these classical programming topics, programming theorems and enumerators can also be implemented with the principles of functional programming we present implementations using the functional programming paradigm along with concepts from object-oriented programming. With these we aim to prove that it is possible to introduce programming theorems and enumerators in introductory programming with functional programming as well. This paper also presents the possibility of a *higher-order-first* approach to programming education and the possible educational advantages this method.

Keywords: introductory programming, programming theorems, functional programming, object-oriented programming, enumerators, higher-order-first approach

1. Introduction and state of the art

In introductory programming the choice of first programming language and the corresponding first programming paradigm has a very important role. Based on the foundation of programming paradigms several strategies for teaching introductory programming exist, such as the algorithm-first, imperative-first, functional-first, hardware-first, etc. approaches [1]. In Hungarian literature a slightly different naming convention is present [2] but the core concepts are basically the same.

Based on our investigation of the curricula of the most renowned universities in Hungary that have some form of Computer Science (CS) of Computer Engineering (CE) program[1], in Hungarian practice an algorithm-first approach is used in most cases starting with basic input-output operations and basic control structures and algorithms. This is usually followed up by the learning about the object-oriented paradigm. Some universities also introduce the functional paradigm early, along with the imperative and the object-oriented paradigms. The training programmes of the top US and European universities with CS or CE programs[2] follow a similar pattern, however courses on functional programming are more prevalent. This shows that alongside the usual imperative-first and algorithms-first approaches [3] the functional-first approach is also relevant [4,5] in current practice.

The goal of this paper is to show that it is possible to use the tools of functional programming to introduce various topics of the classical algorithm-first and object-oriented-second methodologies, namely programming theorems and enumerators, and to show the analogies between the classical

[1] Eötvös Loránd University, University of Debrecen, Budapest University of Technology and Economics, University of Szeged, University of Pécs

[2] Stanford University, MIT, Carnegie Mellon University, Harvard University, University of Oxford, ETH Zurich, University of Cambridge, Imperial College London

and functional implementations. Based on the findings of this investigation we also present a possible direction for creating a new methodology for early programming education.

2. Programming theorems

Classic Hungarian programming education primarily uses the algorithm-first, object-oriented-second approach based on the determinative works of Szlávi and Zsakó [6] that were also published in the booklet series Mikrológia [3,7,8]. One of the core concepts of this so called "Systematic programming" methodology is the usage of programming theorems. Programming theorems are a group of basic algorithms that provide solutions to generic group of algorithmic problems [7,9,10,11]. They also provide a framework for problem-solving by making the specification, planning and implementation of algorithms straightforward by using an analogy-based approach [10]. Szlávi and Zsakó identify eleven such programming theorems [7,11], the system of Gregorics has seven [9,10]. While there is a large intersection between the two lists, the names of theorems can vary. There is also a third source of naming conventions to consider: the vocabulary (keywords) of programming languages. In this paper we follow the naming conventions used by most programming languages to refer to the programming theorems.

All programming theorems can be defined as a function that takes one or more collections of values as input. The type of the values in these collections can be arbitrary. Hereinafter we will use the generic types `X` (and `Y`) to refer these types. Szlávi and Zsakó group the programming theorems into two categories based on the number of their inputs and outputs [7,11] Based on this categorization they identify the following groups:

In the first category there are programming theorems that take a single collection of values as an input and output a single value:

- **Reduction (also called Folding):** Returns a single value by applying a combining function systematically to a cumulative value and each element in the input collection. The combining function takes two values of type `X` and produces a single value of type `X`.
- **Maximum Selection**: Returns a single element (or its index) from the input collection that is the maximum (or minimum) based on some comparison between elements.
- **Counting:** Returns a single integer value that is the number of elements in the input collection that meet a certain condition.
- **Decision:** Returns a single logical value that shows whether the input collection has any values that meet a certain condition.
- **Selection:** Returns the first element (or its index) from the input collection that meets a certain condition (existence of such item is guaranteed by the precondition).
- **Search:** Returns the first element (or its index) from the collection that meets a certain condition if there is any such element.

The second category has programming theorems that take one or more collections of values as an input and output one or more collections of values:

- **Mapping:** Returns a new collection in which each element is a transformed version of the value of the element with the same index in the input array. The transformation is done by some transforming function that takes a value of type `X` and returns a value of type `Y`.
- **Filtering:** Returns a new collection that contains those and only those elements in the input collection that meet a certain condition.

- **Partitioning:** Returns two collections. The first contains elements from the input collection that meet a certain condition, the second contains those elements that do not.
- **Intersection:** Takes two or more sets with elements of type `X` and returns a new set that contains those elements of the input sets that are present in all of them.
- **Union:** Takes two or more sets with elements of type `X` and returns a new set that contains all values that are present in any of the input sets.

2.1. Imperative and functional implementations

In classical algorithm-based programming we use the core concepts of imperative programming (i.e. statements, loops, and conditions) to implement programming theorems. In the following sections we provide the formal signature for each programming theorem and two implementations for each: the classic implementation using imperative programming and an implementation using functional programming with the concept of folding. The goal of this comparison is to prove that the programming theorems can be introduced through not only imperative but functional programming as well.

Notes for the implementations

For the following code examples a programming language or algorithm/function-description method had to be chosen [1,2,12,13]. For the purpose of the demonstration we chose the TypeScript programming language[3] as it is a multi-purpose programming language that supports many programming paradigms (including imperative, functional, and object-oriented); thus, we can use it in all examples. This helps with the transition between the concepts that we present in the following sections.

The syntax of TypeScript follows the conventions of C-style programming languages making the code easier to read. TypeScript also has support for static type annotations and template functions and classes. This makes the following code snippets generic for any input and output value types. All the implementations below are written as template functions and classes (using generic types `X` and `Y`). Another reasoning for choosing the TypeScript programming language for the demonstration is that TypeScript (and the JavaScript language that it is a superset of) has a lot of properties that are beneficial from an educational point of view. [14,15]

The TypeScript programming language has a special value called `undefined` that represents a missing on uninitialized value. We used this `undefined` value for the Search theorem to represent "no output". Implementations of some programming theorems assume a non-empty array as an input. Checking for this precondition is not present for more concise code. We also used some aliases for built-in types and values for the same reason. The list of these aliases can be seen in Table 1. For the functional implementations we use the array destructuring[4] (`[x0, ...xs]`) and conditional statement (`a ? b : c`) features of TypeScript. All functional code snippets with the exception of the Selection theorem are based on pattern matching with conditional statements: *if the input collection has exactly one element/has no elements then return some value, otherwise use a recursive formula to calculate the result*. If a programming theorem cannot accept an empty input, then the first check

[3] https://typescriptlang.org

[4] https://www.typescriptlang.org/docs/handbook/variable-declarations.html#array-destructuring

is whether there is only one element left in the collection. If an empty input collection is accepted, then we check whether the first element is `undefined` (i.e. the input is empty).

Alias (symbol)	Meaning	TypeScript value/type
Z	integer type	`number` (type)
R	real type	`number` (type)
B	logical type	`boolean` (type)
T	true logical value	`true` (value)
F	false logical value	`false` (value)
U	undefined type and value	`undefined` (type, value)

Table 1: Aliases used in the code examples

Reduction, Counting, Maximum Selection

```
type ReduceTheorem  = <X> (x: X[], f: (s: X, e: X) => X) => X
type MaximumTheorem = <X> (x: X[], m: (a: X, b: X) => X) => X
type CountTheorem   = <X> (x: X[], p: (e: X) => B)       => Z
```

Function signatures of the Reduction, Counting and Maximum Selection theorems

The Reduction (a.k.a. Folding, Sequential computing [11] or Summation [10]) programming theorem has two variants. The first takes the first value of the input collection and uses it as the starting value for the cumulation. The second variant uses a starting value (often named `f0`) for the reduction. This second version is more general but, in most cases, we use a value that is neutral for the `f` function so that the starting value does not change the result (e.g. 0 for addition, 1 for multiplication). This means that most of the time this starting value can be omitted, and we can start the reduction with the first element of the input.

In the following example we show the implementation for the first variant of the Reduction theorem that does not use the starting value. This means that only non-empty inputs can be accepted for the theorem. The Reduction theorem starts with the first element of the input then applies the `f` function to the current result and the next element of the collection until all elements of the input have been processed.

```
reduce: ReduceTheorem = (x, f) => {
  let s = x[0];

  for (let i = 1; i < x.length; i++) {
    s = f(s, x[i]);
  }

  return s;
}
```

```
reduce: ReduceTheorem = ([x0, ...xs], f) =>
  xs.length === 0 ? x0
  :                 f(reduce(xs, f), x0)
```

Imperative and functional implementations of the Reduction theorem

The Maximum Selection theorem [10,11] takes the function `m` as an input which is a function that returns the maximum of two values based on some ordering. The theorem then uses this function to calculate the "larger" value of the current maximum and the next element in the input. This

concept is the same as the Reduction theorem where the `f` function of the reduction is the `m` "maximum" function.

```
maximum: MaximumTheorem = (x, m) => {
  let s = x[0];

  for (let i = 1; i < x.length; i++) {
    s = m(s, x[i]);
  }

  return s;
};
```

```
maximum: MaximumTheorem = ([x0, ...xs], m) =>
  xs.length === 0 ? x0
  :                 m(maximum(xs, m), x0)
```

Imperative and functional implementations of the Maximum Selection theorem

The Counting theorem's [10,11] implementation is based on the concept of adding ones and zeroes for each element in the input based on whether they meet the `p` condition.

```
count: CountTheorem = (x, p) => {
  let s = 0;

  for (let i = 0; i < x.length; i++) {
    if (p(x[i])) {
      s += 1;
    }
  }

  return s;
}
```

```
count: CountTheorem = ([x0, ...xs], p) =>
  x0 === U ? 0
  :          (p(x0) ? 1 : 0) + count(xs, p)
```

Imperative and functional implementations of the Counting theorem

Decision, Selection, Linear Search

```
type DecideTheorem = <X> (x: X[], p: (e: X) => B) => B
type SelectTheorem = <X> (x: X[], p: (e: X) => B) => X
type SearchTheorem = <X> (x: X[], p: (e: X) => B) => X | U
```

Function signatures of the Decision, Selection and Linear Search theorems

The Decision theorem [11] is one of the theorems that is missing from Gregorics' list [10]. It takes the predicate `p` as an input and uses it to decide whether there is any element in the collection that satisfies `p`. Both the imperative and the functional (due to lazy evaluation) implementations stop the evaluation if a "good" value is found.

```
decide: DecideTheorem = (x, p) => {
  let i = 0;
```

```
  while (i < x.length && !p(x[i])) {
    i++;
  }

  return i < x.length;
};
```

```
decide: DecideTheorem = ([x0, ...xs], p) =>
  x0 === U ? false
  :          p(x0) || decide(xs, p);
```

Imperative and functional implementations of the Decision theorem

The Selection [10,11] and the Search [11], (aka. Linear Search [10]) theorems are very similar. Both take a predicate `p` as an input to find an element in the input that satisfies `p`. The main difference is that for the Selection theorem we have an extra precondition: there is at least one element for which `p` is true. That is why we do not have to check if we reached the end of the collection.

```
select: SelectTheorem = (x, p) => {
  let i = 0;

  while (!p(x[i])) {
    i++;
  }

  return x[i];
};
```

```
select: SelectTheorem = ([x0, ...xs], p) =>
  p(x0) ? x0 : select(xs, p);
```

Imperative and functional implementations of the Select theorem

The Search theorem does not have this precondition. Some implementations return a logical value that indicates whether we found any element for which `p` is true. Other implementations return a special value for inputs without a "good" value. In our implementation we return `undefined` if there are no element in the input for which `p` is true.

```
search: SearchTheorem = (x, p) => {
  let i = 0;

  while (i < x.length && !p(x[i])) {
    i++;
  }

  return i < x.length ? x[i] : U;
};
```

```
search: SearchTheorem = ([x0, ...xs], p) =>
  x0 === U ? U
  : p(x0)  ? x0
  :          search(xs, p);
```

Imperative and functional implementations of the Search theorem

Mapping, Filtering, Partitioning

```
type MapTheorem       = <X, Y> (x: X[], f: (e: X) => Y) => Y[]
type FilterTheorem    = <X>    (x: X[], p: (e: X) => B) => X[]
type ParitionTheorem  = <X>    (x: X[], p: (e: X) => B) => [X[], X[]]
```

Function signatures of the Mapping and Filtering theorems

The Mapping theorem (aka. Copying [11]) has an additional generic type `Y` present in its function signature. The reason for this is that the `f` function may transform values to a different type than the type of the input values (e.g. mapping strings to their lengths). This theorem's implementation is based on either the option to add a new element to an array (`push`) or the ability to construct a new array by listing the elements of another array using the spread operator (`...`).

```
map: MapTheorem = (x, f) => {
  let s = [];

  for (let i = 0; i < x.length; i++) {
    s.push(f(x[i]));
  }

  return s;
};
```

```
map: MapTheorem = ([x0, ...xs], f) =>
  x0 === U ? []
  :          [f(x0), ...map(xs, f)];
```

Imperative and functional implementations of the Map theorem

The Filtering theorem (aka. Multiple Item Selection [11]) uses a similar concept as the Mapping theorem. The difference is that instead of changing the values of the input collection when we copy it to the output collection, we may omit some elements based on the predicate `p`. It is also similar to the Search theorem with the exception that it does not only search for the first value that satisfies the `p` condition but creates a collection of all of such elements in the input.

```
filter: FilterTheorem = (x, p) => {
  let s = [];

  for (let i = 0; i < x.length; i++) {
    if (p(x[i])) {
      s.push(x[i]);
    }
  }

  return s;
};
```

```
filter: FilterTheorem = ([x0, ...xs], p) =>
  x0 === U ? []
  : p(x0)  ? [x0, ...filter(xs, p)]
  :          [...filter(xs, p)];
```

Imperative and functional implementations of the Filter theorem

The classic imperative implementation of the Partitioning theorem [7,11] is just a more efficient algorithm for applying the Filtering theorem twice. It uses a single loop for both filters thus making the algorithm faster.

Unlike other theorems Partitioning returns two values (two collections) as an output. In the implementations below we used a TypeScript array to return both collections at the same time.

```
partition: ParitionTheorem = (x, p) => {
  let s1 = [], s2 = [];

  for (let i = 0; i < x.length; i++) {
    if (p(x[i])) {
      s1.push(x[i]);
    } else {
      s2.push(x[i]);
    }
  }

  return [s1, s2];
};
```

```
const partition: ParitionTheorem = ([x0, ...xs], p) =>
  x0 === U ? [[], []]
  : p(x0)  ? [[x0, ...partition(xs, p)[0]], [...partition(xs, p)[1]]]
           : [[...partition(xs, p)[0]], [x0, ...partition(xs, p)[1]]]
```

Imperative and functional implementations of the Filter theorem

Intersection, Union

```
type UnionTheorem     = <X> (x: X[], y: X[]) => X[]
type IntersectTheorem = <X> (x: X[], y: X[]) => X[]
```

Function signatures of the Partitioning, Intersection and Union theorems

Even classically the Intersection and Union theorems are just combination of other theorems [7,11]. The Intersection theorem can be viewed as the combination of the Filtering and the Decision theorems, while the Union theorem is basically a Mapping theorem plus the combination of again Filtering and Decision. For this reason, we do not detail the implementations for these theorems.

2.2. Programming theorems with higher-order functions

In addition to using *folding* and *recursion*, another way of implementing programming theorems with functional programming would be by using basic *higher-order functions*. The higher-order functions `reduce`, `map`, and `filter` are present in practically every programming language that supports functional programming (names may vary). Using only these three higher-order functions, it is possible to create easy "one-liner" solutions for all programming theorems (see figure below). This means that introducing only the concept of higher-order functions and these three basic functions are enough to easily solve problems that require the usage of programming theorems.

```
reduce      = (x, f) => x.reduce(f)
count       = (x, p) => x.filter(p).length
maximum     = (x, m) => x.reduce(m)
decide      = (x, p) => x.filter(p).length > 0
select      = (x, p) => x.filter(p)[0]
search      = (x, p) => x.filter(p)[0]
map         = (x, f) => x.map(f)
filter      = (x, p) => x.filter(p)
partition   = (x, p) => [x.filter(p), x.filter(e => !p(e))]
```

Functional implementation of the theorems with higher-order functions

As seen in the code snippet above, `reduce`, `map`, and `filter` are exactly equivalent with the corresponding Reduction, Mapping and Filtering programming theorems. All the other theorems can be implemented using only these three higher-order functions. Many functional programming languages provide built-in functions for more than only these three theorems (e.g. in TypeScript the `some` method for the Decision and the `find` method for the Selection and Search theorems). However, `reduce`, `map`, and `filter` are very common in real-life programming and are enough to provide an "easy-enough" solution to the rest of the theorems.

Szlávi and Zsakó group programming theorems based on whether their output is a single value or one or more collections of values [11]. Another way to group them could be based on their form in functional programming. It is possible to group the theorems into three distinct categories based on which higher-order function can be used for their implementations.

Reduction	**Filtering**	**Mapping**
Reduction	Filtering	Mapping
Maximum Selection	Decision	
	Searching	
	Counting	
	Partitioning	

Grouping theorems based on their implementations with higher-order functions

It is also possible to implement the functional Mapping and Filtering theorems using the Reduction theorem just like in imperative programming [11,16], however that would result in overly complicated solutions for many theorems.

3. Enumerators

One of the classic ways to follow up an introductory, algorithm-first programming education is to continue with the means of data encapsulation and proceed towards object-oriented programming. This usually leads to the introduction of abstract data structures, classes, and objects. Another approach for proceeding towards object-oriented programming is with *enumerators*. Classic (imperative) enumerators are data types that have the following properties [9,17]:

- It is possible to point to its first element (based on an internal ordering),
- step to the next element,
- ask for the currently pointed element,
- and ask if the enumeration has ended.

The classic, imperative implementations of programming theorems all work with such enumerators. Using the interface of enumerators all the array and indexing specific code in the programming theorem can be easily replaced. [17]

```
interface Enumerable<X> {
  first   : () => void;
  next    : () => void;
  current : () => X;
  end     : () => boolean;
}
```

Interface for "classic" enumerators

```
const reduce = <X>(x: Enumerable<X>, f: (s: X, e: X) => X) => {
  x.first();
  let s = x.current();

  for (x.first(); !x.end(); x.next()) {
    s = f(s, x.current());
  }

  return s;
}
```

Implementation of the Reduction theorem using "classic" enumerators

Collections are a subtype of enumerators that store values of a specific type that can be enumerated [9]. They extend the `Enumerable` interface by a method that allows the addition of an element into the collection. This is required to implement programming theorems that output not only a single value but one or more new collections (e.g. Mapping, Filtering).

```
interface Collection<X> extends Enumerable<X> {
  add : (e: X) => void;
}
```

Interface for collections

3.1. Functional enumerators

It is also possible to create enumerators for the functional implementations of programming theorems as well. This requires creating a new definition for functional enumerators to suit our requirements. The requirements for a functional enumerator are the following:

- It is possible to decide whether it is empty,
- ask for the first (head) element,
- ask for the rest of the elements (tail) – i.e. elements except for the first.

Programming theorems for such enumerators can be implemented as standalone functions or as methods of an enumerator class itself [18]. If we use the latter method, we must use an abstract class instead of an interface to be able to implement the theorems as class methods. As for the methods of the `Enumerable` interface, we leave them as abstract methods, showing that these must be implemented for each specific enumerator (e.g. sequential input file enumerator, range enumerator, etc). The theorems can still use these abstract methods in their implementations.

```
interface Enumerable<X> {
  isEmpty : () => boolean;
  next    : () => X;
  rest    : () => Enumerable<X>;
}
```

Abstract class and methods for "Functional" Enumerators

This functional enumerator interface can also be extended to allow the addition of an element. The signature of this extension to create the `Collection` interface is similar to the method required for classic collections. The main difference is that as in functional programming it is not allowed to change the internal state of an object, we must construct a new object when we add an element to a collection.

```
interface Collection<X> extends Enumerable<X> {
  add : (e: X) => Collection<X>;
}
```

Interface for "functional" collections

3.2. Programming theorems for functional enumerators

Programming theorems for functional enumerators can be implemented either as standalone functions or methods on the enumerator with the folding method that is shown in Section 2.1. The main difference between the two approaches is that if a theorem is implemented as a method then it does not have to take the collection as an input parameter, it is automatically passed via the `this` reference. As shown in Section 2.2, we only need to implement three methods, `reduce`, `map`, and `filter` to have access to all programming theorems.

As with these enumerators we target functional programming it is important that the theorem implementations work in accordance with the main principles of functional programming. This means that none of the theorems can change the internal state of an enumerator (they should be immutable) and must return a new enumerator when necessary.

```
abstract class CollectionWithTheorems<X> extends Collection<X> {
  abstract isEmpty : ()     => boolean;
  abstract first   : ()     => X;
  abstract rest    : ()     => CollectionWithTheorems<X>;
  abstract add     : (e: X) => Collection<X>;

  reduce = (f: (s: X, e: X) => X): X =>
    this.rest().isEmpty() ? this.first()
                          : f(this.rest().reduce(f), this.first());
  map = <Y> (f: (e: X) => Y): MappableCollection<Y> =>
    this.empty()          ? this.constructor()
```

```
                          : this.rest().map(f).add(f(this.first()));
  filter = (p: (e: X) => B): FilterableCollection<X> =>
    this.empty()          ? this.constructor()
    : p(this.next())      ? this.rest().filter(p).add(this.first())
                          : this.rest().filter(p);
}
```

Abstract class with the signatures and folding-based implementations of programming theorems

4. Educational considerations

In classic programming education we usually follow an algorithm-first, object-oriented-second approach. With this method the focus in early programming is on the *low-level* concepts, i.e. *how things work* and how to implement basic algorithms. With a functional-first approach the same can be said if we start with the *low-level* concepts first, like pattern-matching or folding. This approach results in a *bottom-up* learning process.

However, there is the possibility to start programming education with a *higher-order-first* approach. This would mean that programming is introduced with *high-level*, functional-style programming: using collections and the `reduce`, `map` and `filter` functions as shown in Section 2.2. This would allow students to easily solve most data-processing tasks easily using high-level tools. Based on our experiences in various introductory and web programming courses and our experience with teaching pupils in summer camps [15] this *top-down* learning approach can emphasise *problem-solving,* giving student early satisfaction and quick success to keep their motivation high. Principles of object-oriented programming could also be introduced early by data-encapsulation with classes and objects and data-processing methods. It would also be possible to use this approach combined with web technologies, web programming in the browser, and the principles of the constructionist learning theory to create a motivating and efficient framework for learning programming [15].

In later stages of a *higher-order-first* educational approach, it could be possible for students to learn about the internal operation of the functional theorems by creating custom enumerable data structures. The implementation of the theorems on custom enumerators could be either in a functional or imperative style thus focusing on *how things work*. This approach could help students to learn about various programming paradigms and how to combine them.

As such *higher-order-first* approach would initially only use the Reduction, Mapping and Filtering theorems as shown in Section 2.2, some other theorems will be less efficient than some other implementations. One example would be the Partitioning theorem that is solved by applying the Filtering theorem twice in succession (thus iterating through the input two times), however on a lower level it can be solved by a single iteration over the input collection. In a top-down learning approach this lack of efficiency is not necessarily an issue, as the focus is on solving problems. Most of the times we do not require highly efficient programs and working with the occasional sub-optimal theorem implementation is perfectly fine. Also, some programming languages provide many built-in higher-order functions that solve more theorems efficiently similarly to how the `reduce`, `map` and `filter` functions solve the Reduction, Mapping and Filtering theorems. In later stages of the learning process the efficiency aspect can be covered in more detail as well, and more effective solutions can be implemented with imperative or low-level functional programming.

5. Conclusions

Programming theorems and enumerators form a solid foundation for classical *algorithm-first*, *object-oriented-second* programming curriculums that are very popular all over the globe. This approach emphasises understanding the low-level concepts of programming and how to use those concepts to build more and more complex algorithms and data structures to solve problems. It is possible to use functional programming to implement the same programming theorems and enumerators. This means that it is possible to create a *functional-first* programming curriculum that is analogous to this classic method as it uses the same programming theorems and enumerators as its foundation.

With functional-style programming and higher-order functions three programming theorems are enough to provide concise albeit form an efficiency perspective sometimes sub-optimal solutions to the rest of the theorems. Practically every programming language that supports functional programming have these three programming theorems available as higher-order functions out of the box. Based on these functions it could be possible to create a *higher-order-first* curriculum for teaching programming. This approach would facilitate a problem-solving centred learning process and would focus less on the low-level inner workings of programming theorems in the early stages of learning programming. In later stages it could also be possible to work our ways towards implementing new enumerators and other data structures that give a deeper understanding of the underlying algorithms or functional constructs. The browser and web programming combined with a constructionist learning methodology could provide a suitable environment for learning activities using this *higher-order-first* approach, thus they could hold great potential as a platform for learning programming using this methodology.

Acknowledgement

The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013, Thematic Fundamental Research Collaborations Grounding Innovation in Informatics and Infocommunications).

Bibliography

1. Vujošević-Janičić, M., Tošić, D., *The Role of Programming Paradigms in the First Programming courses*, *Teaching of Mathematics*, vol. 11, no. 2, pp. 63–83, (2008).
2. Szlávi, P., Zsakó, L., *Methods of teaching programming*, *Teaching Mathematics and Computer Science*, vol. 1, no. 2, pp. 247–257, (2003), DOI: 10.5485/tmcs.2003.0023.
3. Szlávi, P., Zsakó, L., *Módszeres programozás: Programozási bevezető* (in Hungarian), in Mikrológia vol. 18. ELTE TTK Általános Számítástudományi Tanszék: Budapest, (1994).
4. Joosten, S., van den Berg, K., van der Hoeven, G., *Teaching functional programming to first-year students*, *Journal of Functional Programming*, vol. 3, no. 1, pp. 49–65, (1993), DOI: 10.1017/S0956796800000599.
5. Chakravarty, M. M. T., Keller, G., *The Risks and Benefits of Teaching Purely Functional Programming in First Year*, *Joural of Functional Programming*, (2004), DOI: 10.1017/S0956796803004805.
6. Szlávi, P., Zsakó, L., *Módszeres programozás* (in Hungarian). Műszaki Könyvkiadó: Budapest, (1986).
7. Szlávi, P., Zsakó, L., *Módszeres programozás: Programozási tételek* (in Hungarian). ELTE Informatikai Kar, (2008).
8. Szlávi, P., Temesvári, T., Zsakó, L., *Módszeres programozás: A programkészítés technológiája* (in Hungarian), in Mikrológia vol. 21. ELTE TTK Általános Számítástudományi Tanszék: Budapest, (1994).
9. Gregorics, T., *Programozás 1. kötet Tervezés* (in Hungarian). ELTE Eötvös Kiadó: Budapest, (2013).
10. Gregorics, T., Kovácsné Pusztai, K., Fekete, I., Veszprémi, A., *Programming Theorems and Their Applications*, *Teaching Mathematics and Computer Science*, pp. 213–241, (2019), DOI: 10.5485/TMCS.2019.0466.
11. Szlávi, P., Zsakó, L., Törley, G., *Programming Theorems Have the Same Origin*, *Central-European Journal of New Technologies in Research, Education and Practice*, vol. 1, no. 1, pp. 1–12, (2019), DOI: 10.36427/cejntrep.1.1.380.
12. Kruglyk, V., Lvov, M., *Choosing the First Educational Programming Language*, in *CEUR Workshop Proceedings*, (2012), vol. 848, pp. 188–198.
13. Van Roy, P., Haridi, S., *Teaching Programming Broadly and Deeply: The Kernel Language Approach*, in *IFIP Advances in Information and Communication Technology*, (2003), vol. 117, pp. 53–62, DOI: 10.1007/978-0-387-35619-8_6.

14. Horváth, G., Menyhárt, L., *Teaching introductory programming with JavaScript in higher education*, in *Proceedings of the 9th International Conference on Applied Informatics*, (2015), pp. 339–350, DOI: 10.14794/icai.9.2014.1.339.

15. Visnovitz, M., Horváth, G., *A Constructionist Approach to Learn Coding with Programming Canvases in the Web Browser*, *CONSTRUCTIONISM 2020*, pp. 1–8, (2020).

16. Gregorics, T., *Force of Summation*, *Teaching Mathematics and Computer Science*, pp. 185–199, (2014), DOI: 10.5485/TMCS.2014.0365.

17. Gregorics, T., *Programming Theorems on Enumerator*, *Teaching Mathematics and Computer Science*, pp. 89–108, (2010).

18. Gregorics, T., *Programozás 2. kötet Megvalósítás* (in Hungarian). ELTE Eötvös Kiadó: Budapest, (2013).

Authors

VISNOVITZ Márton

Eötvös Loránd University, Budapest, Hungary
3in Research Group, Martonvásár, Hungary
e-mail: visnovitz.marton@inf.elte.hu

About this document

Published in:

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

Volume 2, Number 2. 2020

ISSN: 2676-9425 (online)

DOI:

10.36427/CEJNTREP.2.2.965

License