Teaching Future Teachers to Code – Programming and Computational Thinking for Teacher Students

E. Kaila*, M-J. Laakso* and E. Kurvinen*
* University of Turku/Department of Future Technologies, Turku, Finland
{ertaka, milaak, emakur} @utu.fi

Abstract - Programming is one of the key assets in the modern society. In addition to teaching programming to future programmers, it has become more and more important to teach it alongside computational thinking to all students. We designed and implemented a programming course aimed for teacher students at our university. The goal of the course was to teach programming, computational thinking and the methods and tools used in programming education. The course was divided into two parts: first, the students were taught basic programming skills by utilizing tutorial-based learning with automatically assessed electronic exercises and play-based programming tasks. In the second part, the students utilized the acquired skills by teaching programming integrated into mathematics to children at elementary school level. The students worked really hard throughout the course, completing more than 90% of all tasks in average. Moreover, the feedback from the tutorial sessions and the practice lessons at schools was mainly positive.

Keywords - programming; computational thinking; teacher education; teacher students

I. INTRODUCTION

In the modern society, a knowledge of programming and computational thinking is quickly becoming skill deemed necessary for all people, not just programmers. Hence, the educators at all school levels need to have adequate knowledge about programming and the methods for teaching it. The obvious way to ensure this is include programming into curriculum of future teachers at teacher education institutes. With this in mind, we designed and implemented a programming course aimed for teacher students at University of Turku. It was designed to give students sufficient knowledge about programming and computational thinking with tutorial-based, automatically assessed exercises. The course was concluded with three practice lessons, where the students collaboratively taught programming (integrated into mathematics) to children at elementary school level. In this paper, the course design and implementation are described along with results and students’ perceptions about the course.

The article is structured as follows: in the following Section, related research is presented. In the third section, we present the course design principles including the learning goals and the methodology designed for the course. In the fourth section, the data collection and research setup are presented followed with results and students’ perceptions. In the final section, the results are discussed and interpreted.

II. RELATED WORK

Programming is a difficult skill to master (see for example [1]). It requires both technical skills and problem solving skills [2], usually addressed as computational thinking nowadays. Several methodologies to teach programming better have been developed or adapted over the years. Such methodologies include for example flipped learning [3] and pair-programming [4]. Vihavainen et al. [5] conducted a systematical review on programming education methodologies, and found out, that the most successful methods were utilization of media computing and group work, to name a few. Automatic assessment [6] is extremely useful in programming education, as programming is a skill that requires continuous training with adequate feedback.

Woodrow [7] presents a study where preservice teachers were taught programming and found out, that their overall attitudes towards computers and computing became more positive. Similarly, Fesakis and Serafeim [8] found out that teaching introductory programming to future teachers by using the Scratch programming environment [8] gained a positive change in attitudes towards programming. Mentz et al. [10] utilized pair-programming to teach programming to teacher students, and found out, that pair-programming can be made more effective if applied with principles of cooperative learning. Bell and Frey [11] included teacher students into their programming camp and also found out, that teaching programming had a positive effect on students’ attitudes towards programming.

III. COURSE DESIGN

The course design started by listing the programming-related skills necessary for the future teachers. The skills were then mapped to three learning goals:

1. Programming skills: although the teachers don’t need to become programmers, basic understanding of programming is still fundamental to teach the topic. The group designing the course has years of experience in developing methodologies for teaching programming to computer science majors (see for example [12][13]). However, the focus, learning goals and the target group in the designed course were a little different. Hence, the...
course was designed to only contain the fundamental programming concepts (such as conditional statements, repetition and functions). Python [14] was decided to be the primary programming language, since it is widely considered as easy to learn but still powerful choice for the first language see for example Granell et al. [15][12].

2. Methodology and tools for teaching programming: one of the main goals of the course designed was to offer concrete tools and methods that can be directly applied to teaching programming in elementary school. Hence, a methodology combining educational technology and hands-on exercises was decided to be designed and applied throughout the course. It was also decided, that the teacher students needed to experiment teaching with the methodology in an authentic setup.

3. Computational thinking: according to Wing [16], computational thinking “will be a fundamental skill used by everyone in the world by the middle of the 21st century”. Computational thinking can be seen as expressing a problem and its solution so that it can be executed, often algorithmically step by step. To illustrate computational thinking as a skill that extends beyond programming, play-based exercises (see for example Leskelä et al. [17]) were decided to be designed.

After the learning goals were established, the course methodology and the tools and materials needed were designed and implemented.

A. Tutorial-Based Learning

Teaching programming in the course was based on the tutorial-based learning methodology. In the methodology, a relatively short lecture is accompanied with a tutorial, where students solve automatically assessed programming exercises collaboratively. The tutorials are answered in a computer lab with course staff present to assist students with their problems. Tutorials were built using educational tool called ViLLE. It is an exercise-based learning tool with focus on automatically assessed exercises with immediate feedback. In ViLLE’s tutorials, the exercises are accompanied with learning material (such as text, code examples, images or videos). An example of ViLLE tutorial is displayed in Figure 1.

However, the materials and the exercises were modified to suit the target group better. A total of seven programming tutorials were prepared with following topics:

1. Introduction to computational thinking and algorithms
2. Introduction to programming concepts (such as succession, variables and expressions)
3. String operations
4. Conditional statements
5. Repetition
6. Procedures and functions
7. Lists (voluntary tutorial)

The tutorial sessions were mandatory (excluding the final one) for completing the course. In addition to tutorials, ViLLE was used for other tasks in the course. The attendances to sessions were recorded with ViLLE using RFID tags delivered to each student. An attendance was automatically registered into ViLLE when the tag was used in the reader. Moreover, feedback was collected from the students after each session. Each feedback survey consisted of the same four questions: “What did you learn?” “What remains unclear after the session?” “How would you improve the lecture or tutorial?” and “How could you utilize this week’s topics in your own teaching?”. Additionally, the students were asked to give a grade of 1 to 5 to the session. The feedback was analyzed after each session, and subsequent lecture material or tutorials modified based on the analysis.

One of the reasons for using ViLLE and the tutorial-based learning throughout the course was to make the students’ progress visible for both, teachers and students themselves. The score limits for different grades were announced at the beginning of the course, and students’ current status was visible for them in ViLLE in real time. In addition, elements utilizing the principles of gamification (see for example Huang et al. [16]), such as virtual trophies, were used.

B. Play-based Exercises

Since computational thinking was one of the key topics in the course, exercises were designed to underline the topic outside of traditional programming. In Leskelä et al. [17] the authors found out, that kinesthetic, play-based exercises can be more useful than graphic programming when illustrating the basic concepts of programming (such as repetition or selection). With this in mind, a set of play-based exercises was designed for the course. Hence, the structure of each three hour session followed the same pattern, illustrated in TABLE 1.
The play-based exercises were done in groups of 2 to 20 students, depending on the nature of the exercise. The exercises were designed for both, teaching programming logic and computational thinking for students, and to provide examples and ideas for their own lessons in the future. Some examples of the exercises used are given below.

1. **Drawing exercise**: one of the students acts as a drawer and the other as an instructor. The drawer’s eyes may be closed. The idea is, that first freely, and then with a limited set of instructions available (such as “pen down” or “turn 90 degrees left”), the pair tries to duplicate the figure given to the instructor. The purpose of the task is to illustrate for example the sequential nature and the limited instruction set of computer programs and the solving of problem algorithmically.

2. **Classification exercise**: the students work in a group of 10 to 20. One of the students comes up with (or is given) basis for classifying the students into two groups. Such criteria could be something like “if person has jeans, (s)he goes to the left group, and if not, (s)he goes to the right group”. The persons classified try to guess the criteria used in classification. There can be rules on how the criteria needs to be expressed (for example by using Python’s if-elif-else conditional structure). The goal of the exercise is to teach selection as a structure and expressing conditional statements validly.

3. **Sorting exercise**: the students are again divided into groups of 10 to 20. The task is to select a sorting criteria (such as age, height or shoe size) and sort the group into ascending or descending natural order based on the criteria. The task can be varied by giving a sorting algorithm which to follow (for example quick sort or merge sort, see [22]) or let the students come up with an algorithm themselves. The goal of the task is to again learn how to solve a problem algorithmically and how to utilize computational thinking in problem solving.

The course staff supervised the exercises and offered hints and tips when necessary. After the exercises, they were shortly discussed with the whole group before advancing to tutorial.

### C. Practice Lessons in Schools

The main goal of the course was to provide students concrete skills, tools and methods for teaching programming and computational thinking in schools. Because of this, it was seen essential to provide the students a possibility to practice the concepts in an authentic environment. Hence, each student participated into three actual lessons held in the elementary schools in the area. The pupils in the participating classes were from grades four to six (in Finnish elementary school 10 to 12 year olds). The students in the course were divided into pairs. Each pair was responsible for one lesson and acted as observer and assistant in two others hosted by another pairs.

In the Finnish curriculum, programming is integrated into other topics. Most often integration is done with mathematics. ViLLE provides an electronic learning path for mathematics (see for example [19][20]), where one lesson a week is transformed into electronic one with automatically assessed math exercises and games. Examples of exercises for math are displayed in Figure 2.

![Figure 2. Examples of math exercises in ViLLE](image)

For the lessons, more programming exercises were integrated within the math exercises. Each lesson started with a play-based exercise, planned by students responsible. The time reserved for the exercise was around 10 to 20 minutes. The rest of the lesson was dedicated to programming and math exercises in ViLLE, and the role of the students (the pair responsible and the observers) was to assist students with problems and questions.

### IV. Research Setup and Data Collection

The course was arranged for the first time in the spring of 2017. The course was mainly targeted for teacher education students in the university, but slots were opened for computer science majors as well. The hope was to create multidisciplinary teaching teams with previous experience from teaching and from programming. A total of 41 students enrolled into the course. Two students did not attend the course despite enrolling, so the final number of students was 39 (30 students from the teacher education department and 9 computer science majors): 22 (56 %) of all students were female.

The course lasted a total of 11 weeks. The first seven weeks were reserved for learning programming and computational thinking using tutorial-based learning and play-based exercises. There was one three-hour session a week with possibility to continue working on the tutorial exercises at home if they were not completed in the session. Weeks 8 to 10 were reserved for the practice lessons in schools and the final week was used for a wrap-up session, where both students and the course staff gave feedback and discussed the course implementation and possibilities for improvements together.

ViLLE collects automatically data from all submissions. In addition to score obtained, it automatically records for example the time used and a
total number of submissions to the exercise. ViLLE was also used to collect the attendance data in all course sessions, including the practice lessons in schools. The students answered to a short survey after each tutorial-based session (see Section 3.1 for details). In addition, the students evaluated all practice lessons via another survey.

V. RESULTS

The attendance to at least five of the seven tutorials and to all three practice lessons was mandatory. The average number of attendances per student was 9.36 out of 11 total. The average number of points collected from each of the tutorials is displayed in Figure 3.

![Figure 3. The average points collected in seven tutorials of the course](Image)

As seen in the figure, the average number of points collected was quite high in the first six tutorials, but dropped a little in the final tutorial (69.7%). This is very likely due to attendance into final tutorial being voluntary. The total average for all tutorials combined was 91.1 %. If the final, voluntary tutorial is omitted, the average is as high as 94.6 %. Hence, it seems that in average the students completed a very high number of all exercises in all tutorials.

In addition to tutorials, the students could collect a minor amount of points from attendances, answering surveys (although the final survey was answered as anonymous, and did not hence add to total points) and by trying out a small set of math exercises in ViLLE before the practice lessons started. The average number of points collected from each section of the course is shown in Table 2.

<table>
<thead>
<tr>
<th>TABLE 2. Points Collected from different sections of the course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attendances</td>
</tr>
<tr>
<td>-------------</td>
</tr>
<tr>
<td>Tutorials</td>
</tr>
<tr>
<td>Surveys</td>
</tr>
<tr>
<td>Additional ViLLE Exercises</td>
</tr>
<tr>
<td>Total</td>
</tr>
</tbody>
</table>

As seen in the table, the grade average was highest on weeks 2 and 4. The lowest average was given to session 5, where the topic was repetition. Although important topic, it was probably more difficult than the previous ones. There were only 21 answers to the seventh weekly survey. This can be explained by the fact that unlike the previous sessions, the seventh session was voluntary. All in all, the grade for all sessions was still towards positive (>3.50).

The answers to open questions were very useful for fine-tuning the course while it was going on. The answers to the first question, “What did you learn in this session?” mostly list out the topics covered and Python keywords (such as “the difference between operator and operand”, “str, len and substring” and “using the while statement”). Although there were references to play-based exercises and lecture in the answers, most of them were about the tutorials. Weekly surveying and reacting to the answers seemed to please students, for example one student answered that “[Today] I learned to use subprograms. Tutorial was a lot easier to comprehend than last week’s tutorial. It is great that you read the feedback and modify the teaching accordingly”.

The answers to the second question, “What remains unclear after this session?” were more diverse. Naturally, there were again direct references to topics, concepts and keywords (such as “find and replace” or “indentation”), but many answers gave more detailed and/or general feedback (“Writing code still feels complicated, since I seem to have forgotten the previous concepts every time” or “The course feels a little short for a newcomer like me to learn these things”). There were also some questions about the practical arrangements (“When are the practice lesson schedules announced?”). Still, there were also positive comments, such as “Nothing [remained unclear] – the tutorial was good”.

With the third question (“How would you improve this session?”), a lot of useful feedback on course design and the tools used was collected. The students suggested, for example, that “Pairs in tutorials could be randomly selected” and “It would be more useful if the theory and the exercises were given in shorter segments”). The course staff tried to react to all suggestions, when possible, but naturally all the issues could not be fixed during the course. The final question (“How would you adapt these concepts into your own teaching?”) was mostly answered by listing either the play-based exercises or ViLLE
The students also evaluated all three practice lessons at elementary school level. The students gave grade to seven individual fields of the session. Additionally, they gave an overall grade to the whole session. All answers were given in scale of 1 to 5. For the difficulty level, the scale translated into 1 being too easy, 5 too difficult and 3 a perfect level. For other questions, the scale was translated as 1 being the worst grade and 5 the best. The results are displayed in Table 4.

<table>
<thead>
<tr>
<th>Question</th>
<th>L 1</th>
<th>L 2</th>
<th>L 3</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept difficulty level (1: too easy, 5: too hard)</td>
<td>2.71</td>
<td>2.56</td>
<td>2.70</td>
<td>2.66</td>
</tr>
<tr>
<td>Pedagogical value for programming (5 best)</td>
<td>3.91</td>
<td>3.59</td>
<td>4.00</td>
<td>3.83</td>
</tr>
<tr>
<td>Pedagogical value for mathematics (5 best)</td>
<td>3.22</td>
<td>3.15</td>
<td>3.65</td>
<td>3.34</td>
</tr>
<tr>
<td>Value of play-based exercises (5 best)</td>
<td>4.06</td>
<td>3.59</td>
<td>4.26</td>
<td>3.97</td>
</tr>
<tr>
<td>Programming difficulty level (1: too easy, 5: too hard)</td>
<td>2.81</td>
<td>2.83</td>
<td>3.13</td>
<td>2.96</td>
</tr>
<tr>
<td>Math difficulty level (1: too easy, 5: too hard)</td>
<td>3.16</td>
<td>2.96</td>
<td>2.96</td>
<td>3.03</td>
</tr>
<tr>
<td>Students’ ViILLE fluency (5 best)</td>
<td>3.75</td>
<td>3.85</td>
<td>4.04</td>
<td>3.88</td>
</tr>
<tr>
<td>Overall grade</td>
<td>4.06</td>
<td>4.00</td>
<td>4.17</td>
<td>3.08</td>
</tr>
</tbody>
</table>

VI. DISCUSSION

All in all, the course was successful. The students worked hard throughout the course, completing a clear majority (91%) of all exercises throughout the course. The feedback was also mainly positive, the average grades given to sessions were all over 3.50 in scale of 1 to 5. Naturally, there was some variation between the sessions. The lower grade given to some sessions was mainly due to exercises being considered as too difficult. The course staff reacted to all feedback quickly, which can be clearly seen in the subsequent grades. The practice lessons were also well conceived. The average grades given for difficulty level of programming and mathematic concepts taught were very close to perfect 3 (2.96 and 3.03, respectively). There is also a desirable progress in the grade given to students’ ViILLE fluency, as there is a trend towards higher average in the latter sessions. One hidden objective of the lessons was to demonstrate the difficulties in adapting an educational tool in the classroom, and it seems, that the difficulties in the first lesson were overcome in the latter lessons. The feedback from the students seems to confirm this observation.

Teaching programming to teacher students is naturally different than teaching the same topic to computer science majors or engineering students. The students had different attitudes towards programming, teaching and course practicalities than CS majors in general. Most of the difficulties were overcome during the course. Moreover, a joint discussion session with the course staff and students organized at the end of the course was really useful for the future instances, as a lot of very useful feedback was collected. In the course closing survey, the average grade given for course difficulty (1: too easy, 5: too difficult) was 3.11, which can be seen as a very good result. The comments from the students were also mainly positive: students found the course useful, felt that they were given an adequate knowledge about programming and liked the hands-on lessons with actual pupils in elementary schools. The negative comments mainly concerned the difficulty of programming, which is a well-known problem among programming educators.

REFERENCES

Conference on Innovation and Technology in Computer Science Education. ACM, 2017.


