Early Programming Education Based on Concept Building

Jiří Vaníček • University of South Bohemia, Czech Republic • vanicek/at/pf.jcu.cz

> Context • The main context of this study is the shift of programming education from professional development to general education. **> Problem** • The article deals with methods, environments and approaches to teaching programming to everyone. **> Method** • Conceiving programming education as concept building by creating pupils' mental models in selected didactical environments that are constructed to allow pupils to focus on the given problem and, at the same time, to have the structure of a set of similar short tasks of increasing difficulty with the same underlying concept. Design-based research on the evaluation of curricular materials created according to this method. **> Results** • Specified principles of creation of appropriate materials for teaching programming, intervention conducted with these materials and experience from a pilot research study of teaching that contains signals of how difficult it would be to change teachers' minds to make them willing to accept and implement this approach in their teaching. **> Implications** • The article focuses on applying the theory from mathematics education to a different field. The results could be beneficial for programming curricula education creators; a qualitatively new generation of textbooks on programming education for pupils from an early age could be created using this approach. Future research could focus on teachers' beliefs and the changes to these beliefs when teaching programming in this way. **> Constructivist content** • The theory used has its origin in mathematical constructivism and is based on the work of Papert and Hejný. It could bring experience in applying a proven theory originally used in another discipline. **> Key words** • **Computer science, programming education, junior high school, concept building, Scratch**.

Introduction

« 1 » If informatics is, in accordance with Walter Gander (2014: 7), regarded as a part of general education, computational thinking as the new literacy of the 21st century (Wing 2011) becomes a substantial new skill. Many education authorities believe that it can help children develop their competencies in their early years (Kalaš 2010: 9). This is in line with Jerome Bruner's proposition that the foundations of any subject may be taught to anybody at any age in some form (Bruner 1960: 12).

« 2 » If informatics becomes a component of the compulsory curriculum, programming, perceived as a "playground" where various components of computational thinking such as abstraction, algorithmization, decomposition, evaluation, or generalization (Selby & Woollard 2013) and other cognitive functions can be developed, grows in importance. Consequently, we have to ask what approaches and methods to use when teaching programming with respect to the above-stated goals. When looking for teaching methods, we also have to keep in mind the decreasing age at which the teaching of informatics starts.

« 3 » Rather than focusing separately on educating and training professional IT specialists, it may be more practical to focus on how teachers can learn by doing: how learning and teaching of programming can develop teachers' computational thinking as well as that of the pupils. Many approaches to teaching programming favor pupils' activity, active learning, learning by doing, and the construction of knowledge as a result of active creative work, all this with respect to the view that knowledge and knowing are not transmittable. According to Jean Piaget, knowledge is actively constructed by the learner in interaction with the world, so as Edith Ackermann (2010: 2) suggests, it is worth offering opportunities for children to engage in hands-on explorations that fuel the constructive process. This approach is in line with Piaget's view that "children interpret what they hear in the light of their own knowledge and experience," and Seymour Papert's opinion that "knowledge is formed and transformed within specific contexts, shaped and expressed through different media" (Ackermann 2001: 3, 8).

« 4 » Some might expect that this approach within the popular Scratch programming environment (Resnick et al. 2009: 60) can guarantee that a pupil will learn to program in an innovative way. There are reasons to assume that this is not sufficient. A very important factor in the education process is the quality of the teachers, their experience and prior training and education. Also crucial are their willingness and ability to move from a more traditional model of teaching (which emphasizes reproducing and imitating) to an approach that gives pupils more opportunity for creative activities.

« 5 » This article presents the theoretical basis of a concept-building approach and a framework of curricular materials (textbook) for teaching programming at junior-high-school level by teachers who are not very experienced. In the next part I will present research set within the specific context of the Czech educational system and its recent development.[1] These materials were designed and tested with the main goal of preparing teaching materials that meet the

1 | http://www.msmt.cz/uploads/DigiStrategie.pdf

requirements of the newly revised national computing curriculum (NÚV 2018). This curriculum is expected to come into effect in 2021 and contains programming as a compulsory part of educational outcomes at elementary and high-school levels.

Background

« 6 » It is a challenge to implement a constructivist theoretical basis into teaching programming. One possible new way of doing this is to use the framework "5E" (Explore, Explain, Envisage, Exchange, bridgE) used in the ScratchMaths project for elementary programming education in England (Benton et al. 2016: 29). There is also other introductory programming research that includes "targeted tasks" (Grover et al. 2015; Hansen et al. 2016; Meerbaum-Salant et al. 2013).

« 7 » When looking for ways to conceive teaching materials for programming at junior-high-school level it is possible to turn our attention to theories of mathematics. These theories have been considering how concepts are built for several decades. School mathematics has been seeking new ways of teaching since the 1960s. According to criticism by Morris Kline (1973), the experience from the so-called "New Math" made mathematics educators aware that an understanding of mathematics is not only achieved through the content of the teaching, but also by the method of teaching (Hejný 2012: 43).

« 8 » Several theories deal with concept, which is the key term in the procept theory[2] (Gray & Tall 1994: 117) and in scheme-oriented education theory (Hejný, Slezáková & Jirotková 2013: 995). David Tall and Shlomo Vinner introduced the term "concept image," which includes all mental pictures and associated properties and processes (Tall & Vinner 1981: 152). The APOS theory (Action-Process-Object-Schema) explains how an individual grasps (mathematical) concepts by constructing mental actions, processes, and objects and organizing them into schemas (Dubinsky & Mcdonald 2001: 2).

« 9 » The theory of generic mental models (Hejný 1987: 58, 2012: 44) works with the mechanism of cognitive process and helps to analyze pupils' thinking processes and detect sources of pupils' mistakes. The aim of the approach is to decrease the emphasis on mechanically accumulating knowledge without a deeper understanding. According to this theory, the process of constructing a piece of knowledge starts from initial motivation, moves to the construction of isolated models and results in a creation of the so-called "generic models," which Milan Hejný calls "building blocks of learning with understanding" (Hejný 2012: 44). Having been motivated, a person first observes phenomena in which the new concept is present and creates isolated models (Hejný 1987: 59) that are tested in new situations. With enough time and opportunity to get a sufficient number of isolated models in different situations, the person is able to build a generic model that is universal and should work in all known situations because "it is an example or representative of all its isolated models" (Hejný 2012: 45).

« 10 » A concept is not a data point, a single piece of information. It is an abstraction. To grasp a concept, a pupil needs to create as many isolated mental models of this concept as possible, so that she gets experience with specific cases in which the concept is at play. Isolated models allow pupils to make connections in their minds, i.e., they can look for relationships, structure the information, and construct a generic model of the intended concept. This implies a pupil must go through a number of situations in which the concept appears through various prisms and from different points of view. In these situations, the pupil creates a number of models, including what seem like models but are partial and/or incomplete, and models that are surprising (Hejný 2004: 28). The pupil must come across situations in which the concept behaves in a strange, specific, unexpected way in order to see it plastically.

« 11 » Let me illustrate the importance of understanding concepts with several examples. If we want pupils to understand programming in order to build concepts correctly, it is not enough that they be able to develop a program, create an algorithm, or propose a solution to the problem. They must be able to look at the problem from a distance; they need the practical programming skills that they will then apply when solving the situation. So-called "beaver tasks" from the Bebras challenge contest (Dagienė 2017: 23) are situational informatics tasks that simulate everyday life situations and their informatic dimension. In such tasks, pupils plunge into a described situation that they must grasp, they must come to understand the concepts and terms that are used, find an informatics principle that the task is based on, and solve the problem using cognitive and thinking skills. Part of these tasks is algorithmic or programming: tasks that are hard to solve without an understanding of the concepts related to programming.

« 12 » Hejný claims that understanding is more important than skill. A well-constructed mental model helps one grasp a concept correctly and find relationships between this concept and others. An example of diametrically different behavior of various commands in Scratch is the "pen state" or "sprite[3] visibility," which preserves the given state and is valid until the state is changed. This is in contrast to the executive commands of *stamp* or *change costume*. Pupils have problems with understanding these differences. Another example comes from programming robots. It may happen that the "program is standing and the robot is moving," e.g., having carried out the command that makes the engine work, the program waits for the fulfillment of the condition at the input from sensors before carrying out the command to stop the engine. Children presume that when the program is stopping, the robot should not move. These situations are difficult for pupils if they do not understand the mechanism of the given concept well.

« 13 » We can find some similarities to the approach of concept-building theory in the ideas presented by Piaget (1951, 1952) in terms of accommodation and assimilation. The cognitive progress through an isolated to a generic mental model could be seen as similar to SOLO (structure of observed learning outcome) Taxonomy, in moving

2| A procept is an amalgam of three components: a process, object and symbol. A process produces a mathematical object and a symbol is used to represent either process or object.

3| Sprites are pictures on a Scratch computer program screen, objects to be programmed using scripts (instructions) that control them.

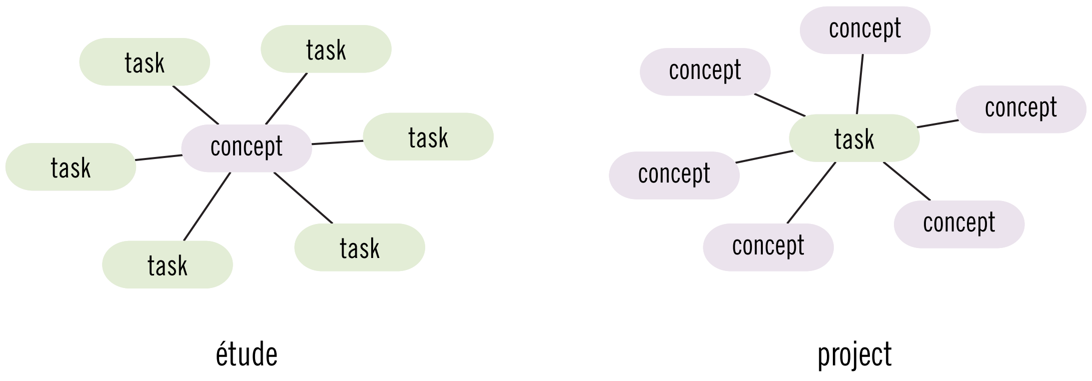


Figure 1 • Comparison of an étude and a project from the point of view of the relationship between the concept taught and the task used.

from unistructural to an extended abstract with particular focus on the relational phase (Biggs & Collis 1982).

Building concepts in études and projects

« 14 » Observations of in-service teachers (who have a lot of teaching experience, but no training in informatics) as well as pre-service teachers (who have an informatics background but no teaching experience) are a rich source of information for how such teachers approach their teaching. I compared approaches to teaching programming used in Czech and other textbooks, and also methodological manuals for the teaching of programming available on the internet. Predominant in these textbooks and manuals are two basic types of programming tasks.

« 15 » One uses a series of several-minute-long programming "études,"[4] always targeting the acquisition of one specific skill or focused on one specific item of knowledge or concept. Pupils can easily check their results and it is easy to organize such lessons, especially for less experienced teachers. But motivation in these activities is weaker than in larger projects.

« 16 » Teaching through larger programming units – the so-called "projects" (as in The LEAD Project 2014), e.g., creating games or stories – often takes the form of a sequence of activities organized as a tutorial or a problem task. It is hard to combine the goal of teaching a specific concept, procedure or method with open-ended activities. These activities cannot target the development of one concept. Programming projects can result in the creation of long multiline codes in which it is harder for the pupil to orient and for the teacher to find the pupil's mistake.

« 17 » In the two ways of selecting types of problems, two different approaches to teaching goals can be observed (see Figure 1): in the case of shorter programming études, the approach is intensive, focused on competencies in the area of concepts, while the other is holistic, more open, emphasizing creativity and focusing on the product (Vaníček 2015: 85).

« 18 » From this point of view, I analyzed learning materials created by the Scratch development team as parts of official Scratch distribution: Tips in Scratch 2.0[5] and Ideas in Scratch 3.0.[6]According to this classification, we can say that these materials are projects. The structure and approach are carried out in tutorials that lead the individual learner step by step. These tutorials tell them exactly what they have to do for the next step in order to finally create some kind of story, tool or game. Each of the presented activities requires them to use dozens of different skills at the same time, and a lot of different knowledge (which is very hard to attain for beginners). This gives the final effect of a finished project, and for the authors of these materials, the introduction of the features of the environment is more important than pupils' understanding of the programming concepts used in the project.

Didactical environments for programming

« 19 » One of the disadvantages of block-oriented programming environments is that users can see all (or most of) the blocks of the language. However, pupils cannot master all of these commands, especially if they are to understand them in depth. Then it becomes difficult to give pupils a problem because if they do not immediately know how to solve the task they may, instead of thinking, waste time by looking for an appropriate tool or command to solve the problem. A tutorial is not a solution because following fully-specified procedures does not help a pupil learn to generalize.

« 20 » Particularly in introductory activities for teaching programming, I find it most efficient to create didactical environments. A substantial mathematics environment (Wittmann 2001: 2) is a set of linked situations that provide problems that allow a pupil to discover important ideas. Hejný added three more conditions to this concept: the motivating power, long-term commitment, and flexibility in the level of difficulty. (Hejný, Slezáková & Jirotková 2013: 998).

« 21 » Mathematics educators use ways of creating learning environments based on one task. Bernd Wollring mentions that task is a subset of task format, and task format

4 | http://www.ucl.ac.uk/ioe/research/projects/scratchmaths/curriculum-materials

5 | https://scratch.mit.edu/download/scratch2
6 | https://scratch.mit.edu/ideas

is a subset of learning environment.[7] In his scheme-oriented approach to mathematics education, Hejný uses learning environments that are close to children's everyday experience e.g., Spider web, Snakes, Stepping, Seat in the bus (Hejný, Slezáková & Jirotková 2013), Father Woodman, Staircases, (Hejný 2012) to create learning environments; these are microworlds in which it is possible to grasp the rules quickly and to solve a given set of similar or progressive tasks. Similar environments have been used in programming education for years (e.g., turtle graphics, Karel the robot). Scratch enables one to create didactically strong learning environments viable as Scratch projects that a pupil opens and works in. Such environments, in which sprites and their costumes are prepared and positioned, the backdrop of a stage is set, and blocks necessary for solving the task are earmarked, serve as task settings in which pupils can concentrate their focus on "putting blocks together." This saves time and could help pupils to avoid errors that are not related to programming, but a lack of understanding of the environment of Scratch (e.g., when a sprite moves in one direction when it looks like it is directed to another, or which part of a sprite body draws a line when using a pen tool).

« 22 » To paraphrase Hejný's characteristics of effective mathematics teaching (Hejný 2012: 44), effective teaching of programming is characterized by three cognitive goals:

- pupils understand programming, their knowledge is not mechanical;
- pupils are intrinsically motivated to work, they are not frustrated by programming;
- pupils develop intellectually, which primarily means the development of the ability to: 1. communicate both orally and in writing, 2. cooperate in a group or even lead a group to solve problems, 3. analyze a problem situation, 4. effectively solve problems, 5. correct one's own mistakes.

7| "Zur Kennzeichnung von Lernumgebungen für den Mathematikunterricht in der Grundschule," University of Bayreuth, 2007. http://www.sinus-transfer.de/fileadmin/MaterialienIPN/Lernumgebungen_Wo_f_Erkner_070621.pdf

Research questions

« 23 » The approach to mathematics teaching that is oriented towards pupil activity by the stimulation of the creation of mental models of mathematics concepts has led us to the question of whether we can use this approach and apply it to programming education from an early age. More specific questions arise:

- Is it possible to create a curriculum for programming teaching that is primarily oriented towards concepts building?
- How can we implement didactical environments so that they take into account the requirements of the method and character of programming software?
- Which types of activities support this approach?
- Which problems arise through the implementation of this approach by experienced teachers and novice teachers in programming teaching?
- Which part of the implementation of this approach creates more obstacles for teachers?

Aims and contribution

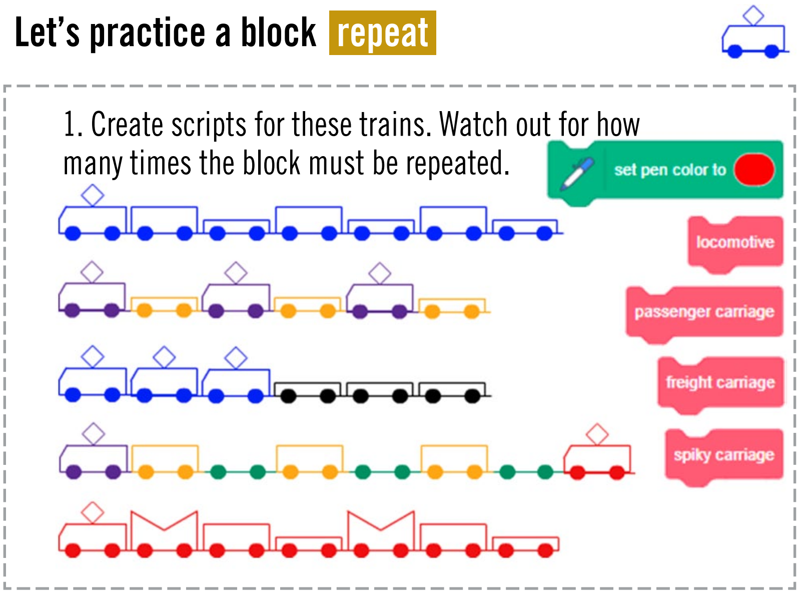
« 24 » The question is how to put into action practical curricula of programming based on a methodology taken from the didactics of mathematics. Although these school subjects are close to each other, the character of teaching them is different. I decided to implement the main change of using études and also creating didactical environments. As basic building blocks of curricula, études allow us to take advantage of the programming language Scratch, so that they meet Papert's requirement called "low floor" (get started easily). Didactical environments, with their variety of carefully prepared activities, could take advantage of Scratch software in another way: "wide walls, which support many different types of projects so that people with many different interests and learning styles can become engaged" (Resnick 2009: 63, 2017).

« 25 » In this part, I define the properties of a curriculum based on building concepts and illustrate them in several particular tasks in a didactical environment. I describe building a curriculum of learning programming with the aim of making universal mental models of programming concepts carefully ordered according to their complexity. I also describe several types of children's activities used in this approach.

Properties of curriculum based on building concepts

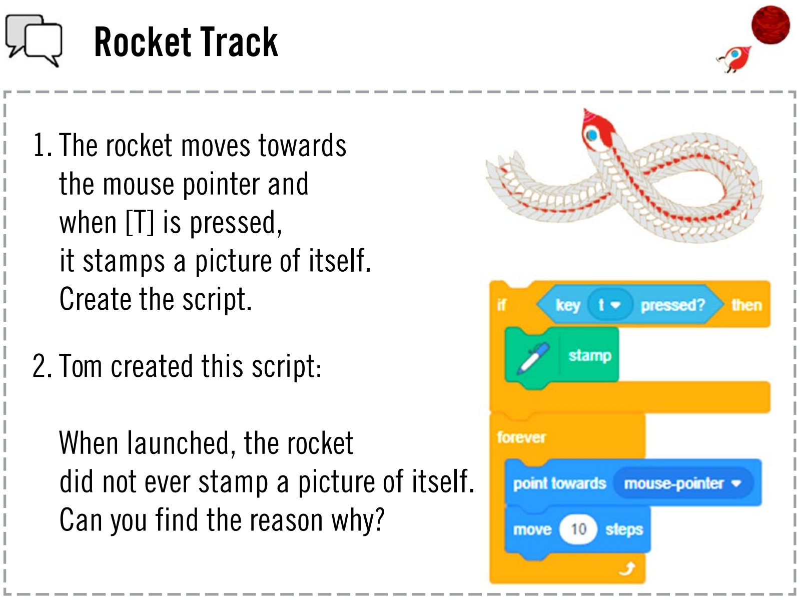
« 26 » The theoretical background described above allows us to state principles for creating a concept-building programming education curriculum.

- It is made up of a lot of very short activities that are related to the same concept. The aim is for pupils to get an idea of the behavior of the given concept in various situations. These short activities help the teacher; unlike in broader activities in which a teacher can get lost, the teacher is able to find a pupil's mistake in a script, and there is less chance that the pupil will program a situation in which the teacher is not able to test, fails to correct, or cannot guide the pupil to discovery.
- Several didactical environments must be used for teaching one concept. These environments must be based on different activities of sprites, e.g., drawing lines, stamping shapes, moving on the stage and changing costumes. In these environments, the intended concept is shown in various situations, allowing the pupil to use abstraction in order to avoid incidental or irrelevant properties and attach properties to the concept that are substantial and independent of a particular environment. The pieces of knowledge gained by the pupil do not remain isolated.
- Mistakes are not unwelcome; on the contrary they are one of the main sources of knowledge. A pupil must be presented with various possible mistakes that can be the result of misunderstanding and that can help the pupil grasp the concept more deeply. Pupils find it motivating if such activities are worded as if somebody has made a mistake in their program and is facing a situation in which they desperately need help. The pupil's task is to figure out what situations expose the mistake, what it is caused by, if it generally is a mistake (some solutions


Let's practice a block repeat

1. Create scripts for these trains. Watch out for how many times the block must be repeated.

Figure 2 • Set of études of an increasing complexity of the used structure of blocks (original material is in Czech language).


Rocket Track

1. The rocket moves towards the mouse pointer and when [T] is pressed, it stamps a picture of itself. Create the script.

2. Tom created this script:

When launched, the rocket did not ever stamp a picture of itself. Can you find the reason why?

Figure 3 • Discussion leading to finding a mistake in the program.

may be erroneous only from certain points of view) and whether the mistake is covert (e.g., the solution is not general enough or the mistake appears only after a change of input parameters or when run again).

- Activities should aim at various programming competencies. Pupils are asked not only to create and run a script but also to read it, interpret it, predict the output, test its correctness, set the situation in a way that would make the mistake manifest, or adapt a working script to solve a similar problem. In this way the pupil's pieces of knowledge do not remain isolated.
- The curriculum should be built on concepts in order of increasing difficulty. According to my long-term experience and after studying other textbooks, I set the order command (block), program (script), sequence (arrangement of blocks), repetition, procedure (new block), event, object (sprite), message, condition, decision-making, arrangement and embedding of structures, parameter, variable (the words in brackets adhere to the terminology used in Scratch).
- When creating a script, either individual work or group work alone is not enough. Activities should be varied with respect to the form of pupils' work. Pupils also need to experiment with various blocks, changes of order or different arrangement of blocks, and different inputs. Some other good activities are individual creation, discussion with justification and reasoning, commenting on other people's work, or unplugged activity in which the pupil in the role of a sprite acts out the demands of the script.
- The curriculum should be graded from basic to more difficult tasks (and to challenging ones for more advanced or faster pupils). The curriculum should have a spiral structure. The mastered concepts re-appear in tasks in later chapters, this time in new relationships to other concepts and in new environments. Therefore, pupils learn by revisiting.
- Each activity, however short, must have some effect, and must provide a new experience to the learner. The pupil becomes used to receiving feedback from the computer and not from the teacher. She anticipates the reaction of the computer. The curriculum respects that the goal of programming from the pupil's point of view is to create something that works with "one click," whether this be a story, game, drawing or piece of music. Pupils' interest and success results in the teachers' revision of beliefs and a newfound willingness to invest in changes in their teaching style.

Illustrations

« 27 » Let me illustrate the previously defined principles in several particular tasks in a didactical environment that was used when creating curricular materials for basic programming for 12-year-old pupils.

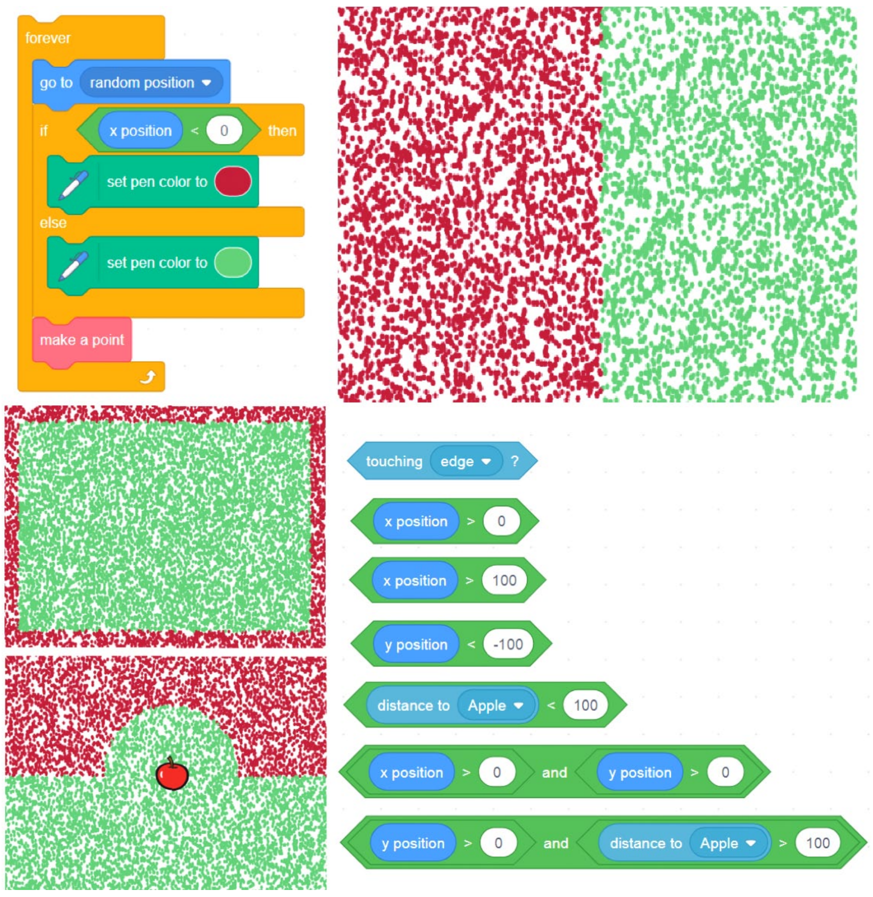
« 28 » The didactical environment "Train" uses Logo-like turtle graphics; each block draws one carriage. Figure 2 shows a set of études for beginners to help them understand how to insert blocks into a loop. Solutions require an increasingly complex structure of blocks in a loop. Tasks constructed in this manner give pupils more points of view of the given concept and provide experience allowing them to build a generic model of the concept of repetition.

« 29 » Figure 3 shows a task in which pupils are asked to find a mistake in the script. Pupils know that the *pen down* command sets the sprite's state to draw a trace until it receives the command *pen up*. For pupils who imagine that the *stamp* command behaves in the same way (makes the sprite continue stamping through subsequent moves), the sprite behaves contrary to their expectations. Pupils are expected to reason and test hypotheses. If more tasks of this type (different kinds of mistakes with different sources) are used, pupils are guided to a deeper understanding of the given concept.

« 30 » Figure 4 shows the didactical environment "Coordinates," the goal of which is to help pupils understand conditions, conditional clauses and coordinates. A randomly jumping sprite makes points on the stage. Their color depends on whether the given condition is fulfilled. The task objective is to create a condition in which the stage will be colored correctly when using the condition in the script. In the bottom part, conditions of a growing complexity are presented (showing the result for the first and the last conditions). By experimenting through changing conditions in the script, a pupil may get to understand coordinates in more depth.

« 31 » Figure 5 shows the didactical environment "Words and Letters," in which pupils have to create words by "stamping" letters. Letters are "stamped" when the sprite switches its shape into a picture of the required letter, stamps itself to the background and then moves to the right. In this advanced task, pupils create a procedure (called print) with a text parameter. This is a step forward compared to the previous level, where parameters were not used and it was necessary to create a specific procedure for each printed letter.

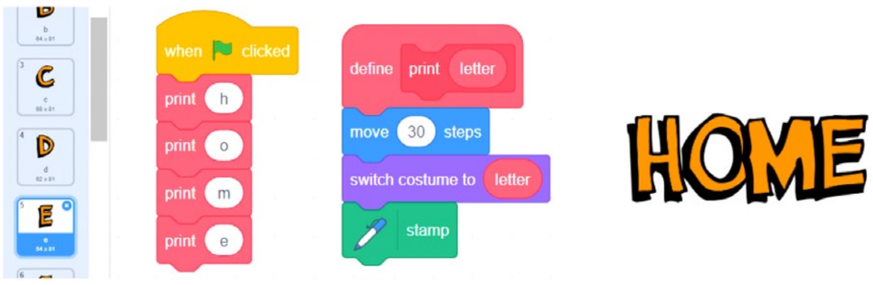
« 32 » Figure 6 shows the activity "Dice game," with random numbers, decision-making and variables for remembering values. Two sprites show a random number and the third sprite in the center decides on the winner. If used repeatedly, wins can be counted and the activity can grow into a simple statistical experiment using the computer as a tool for automatic enumeration (figure on the right).



```
forever
    go to (random position)
    if <(x position) < (0)> then
        set pen color to (red)
    else
        set pen color to (green)
    make a point
```

```
<touching (edge) ?>
<(x position) > (0)>
<(x position) > (100)>
<(y position) < (-100)>
<(distance to (Apple)) < (100)>
<<(x position) > (0)> and <(y position) > (0)>>
<<(y position) > (0)> and <(distance to (Apple)) > (100)>>
```

Figure 4 • Experiment with conditions and coordinates.



```
when flag clicked
print (h)
print (o)
print (m)
print (e)

define print (letter)
move (30) steps
switch costume to (letter)
stamp
```

Figure 5 • Procedures with a text parameter.

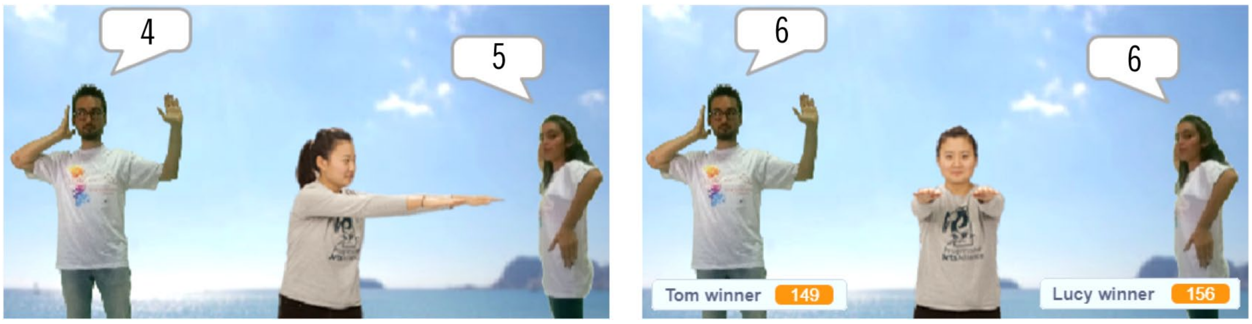


Figure 6 • Dice game.

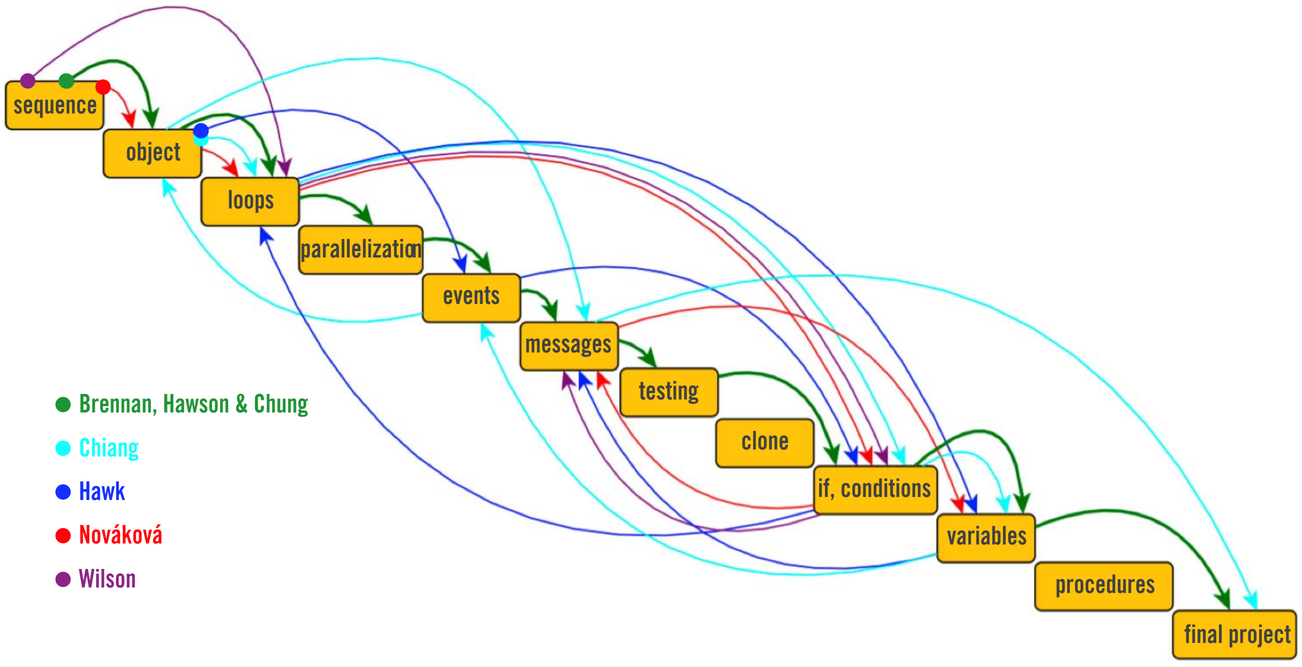


Figure 7 • Order of concepts used in online textbooks for programming in Scratch (see Appendix). Missing concept of procedures in these curricula could be explained by the absence of this tool in the previous version of Scratch 1.4.

Building a curriculum of learning programming using concepts

« 33 » When creating curricular documents for younger classes at high schools based on programming in Scratch, several textbooks and online materials for teaching programming using Scratch were analyzed first (Krejsa 2012: 28). Jan Krejsa and I (as his supervisor) looked for the programming concepts used and the order in which they are taught. The results are shown in Figure 7. Concepts included in national curricular documents for the given age group (NÚV 2018: 12) were also taken into consideration. We follow the line from easy-to-understand concepts to more complex ones. Finally, the chosen order of concepts was most similar to Karen Brennan's materials;[8] we only added procedures and parameters and did not set out a special chapter for concept of object.

« 34 » The constructed curriculum is divided into chapters, each of them related to one of the following concepts:

1. Script (program as a sequence of commands)
2. Cycle (loop with known number of reps)
3. Procedures (subroutines)
4. Conditions (while loop)
5. Events (parallel threads)
6. Objects and communication (sending messages, broadcasting)
7. Branching (if-then command)
8. Coordinates
9. Procedures with parameters
10. Variables

8| http://scratched.gse.harvard.edu/resources/scratch-curriculum-guide

« 35 » My own experience and the study of curricular materials for programming led me to arrange the concepts in order of increasing difficulty and complexity, while making sure that the subsequent concept was not too distant from the prerequisite knowledge. That was why, for example, procedures presented in the form of named sequences of commands were included at a relatively early stage. This allowed us[9] to avoid long confusing scripts that pupils would have difficulties reading. Also, we wanted to make sure that pupils would have enough opportunities to become familiar with procedures by using this concept many times before proceeding to parameters, which is a significant abstract shift.

« 36 » Similarly, the concept of condition is introduced separately to the concept of branching a program. Branching as a concept is only introduced after pupils have mastered the concept of condition they first came across in a while loop. This situation is more favorable for the introduction of the concept of repetition with a condition, because pupils have already grasped the concept of repetition. If we introduced conditions using the command if-then, the situation would be more complicated for pupils because they would have to grasp another concept of branching at the same time as the concept of condition. Both of these concepts are difficult to understand; with their simultaneous introduction there would be a risk that pupils of the particular age group would fail.

« 37 » Each of the concepts is introduced with one basic idea that it should evoke in the pupil. This idea allows the pupil to set it into an everyday context, into their concept map, e.g.:

- script as a set of commands carried out by one click;
- conditions (inside while loop) such as testing when some activity ends;
- events such as testing when some activity begins;
- communication as a way of addressing commands to various addressees;
- parameters as a way of creating a solution for various tasks without modifying the program;
- variables as a way that the computer remembers something.

9| By using the plural pronoun I wish to emphasize that while the conception, main educational ideas and most of the content is my work, my colleagues Ingrid Nagyová and Monika Tomcsányiová collaborated with me creating the educational material and carrying out the pilot study.

« 38 » The idea of the introduced concept should be individually built by each pupil. They should discover it in activities, the goals of which are controlled exploration of the given concept, individual creation and discussion.

« 39 » Pilot testing showed that it was difficult for pupils to build a generic model within the frame of the chapter that introduces the concept. Apart from duration (pupils need enough time to get some experience with the behavior of the concept in various situations), the main reason was that in order to show a particular concept in a specific, more complex situation, pupils first had to be introduced to a concept that was presented later in the curriculum. This meant that even in later chapters it was necessary to go back to some previous concepts and use tasks that were primarily targeting some of the concepts introduced earlier, rather than the main concepts of the particular chapter.

How to make a universal model

« 40 » In this subsection, I want to demonstrate how the curriculum was constructed and I use the concept of repetition as an example. The building of a generic model requires solutions to tasks that show how the concept behaves in isolated situations on a number of isolated models. If a pupil is not able to solve a task based on the behavior of a given concept in a new, dissimilar situation, they have not built the generic model yet. When a generic model is created, the pupil is able to predict the behavior of the concept in a new situation. The strength of the new concept can be tested in more extreme or complex situations that can change the pupil's mental model if it is not yet sufficiently strong.

« 41 » We tried to follow the following procedure of presenting tasks related to a given concept to the learner. However, it was not always possible to reach the final two stages of this procedure:

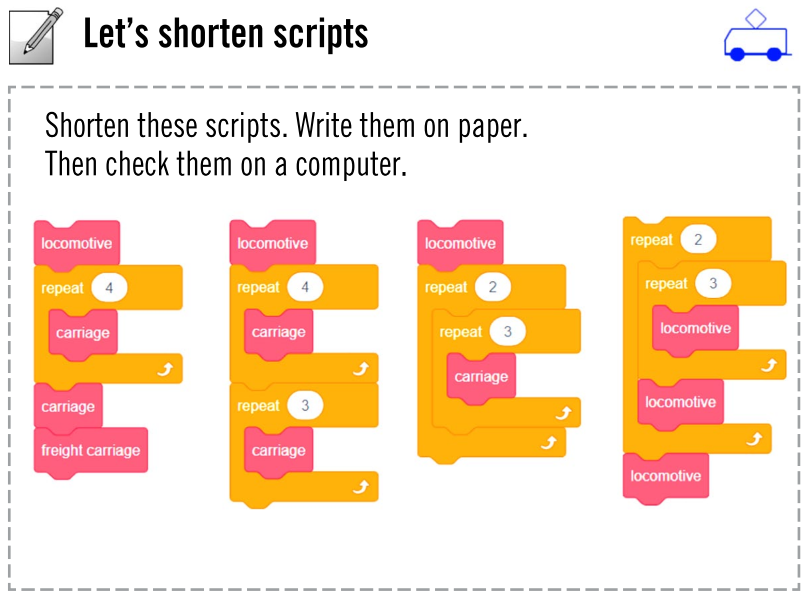
- introducing the pupils to the concept in a situation when the concept is isolated;
- setting the concept into a new, related situation;
- behavior of the context in several situations in which it behaves similarly;
- behavior of the concept in absolutely different but simple situations;
- setting the concept into a complex situation;
- analysis of the given situation with the aim of differentiating the concept from other known, similar concepts;
- thinking about the concept without a specific situation with the aim of abstracting its properties.

« 42 » If this approach is used, the teacher can regulate pupils' learning by choosing suitable activities. If the teacher has a set of activities of increasing difficulty focusing on a given concept available, then if a pupil fails to understand, the teacher may use several variants of the same problem that reinforce the pupil's understanding of the concept before moving on to a more complex task. Thus, the sequence of activities building a generic model is maintained.

« 43 » To finish building a generic model, more complex situations are required. After pupils achieve some level of understanding of the concept (and begin to be more independent), it is possible to replace single études with something like projects, "such as" sets of follow-up études that lead to one target, e.g., the creation of a story, animation or simple game.

« 44 » When creating individual activities targeted at building a generic model, we always constructed a detailed set of skills that the sequence of activities gradually developed. This allowed us to construct the subsequent steps of selection and arrangement of activities. Let me illustrate this with the concept of repetition:

- First, the presentation of a situation in which repetition occurs and where the use of the Repeat command is appropriate;
- introduction of the Repeat command (technique of its creation);
- creation of scripts with repetition of one command (the pupil determines the number of repetitions);
- adding other commands into the script (the pupil decides which commands are



Let's shorten scripts

Shorten these scripts. Write them on paper.
Then check them on a computer.

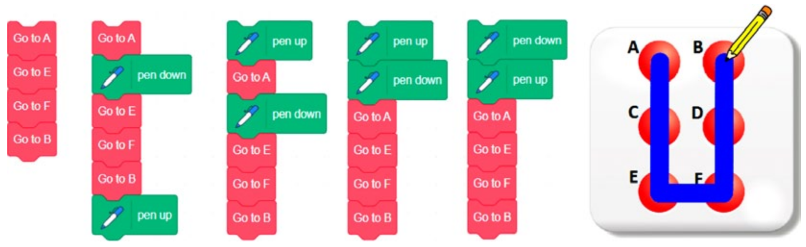
```
locomotive
repeat 4
    carriage
carriage
freight carriage
```

```
locomotive
repeat 4
    carriage
repeat 3
    carriage
```

```
locomotive
repeat 2
    repeat 3
        carriage
```

```
repeat 2
    repeat 3
        locomotive
    locomotive
locomotive
```

Figure 8 • Optimization of a code as a task that develops thinking about a concept without a specific situation.



```
Go to A
Go to E
Go to F
Go to B
```

```
Go to A
pen down
Go to E
Go to F
Go to B
pen up
```

```
pen up
Go to A
pen down
Go to E
Go to F
Go to B
```

```
pen up
pen down
Go to A
Go to E
Go to F
Go to B
```

```
pen down
pen up
Go to A
Go to E
Go to F
Go to B
```

Figure 9 • Discussion activity in the didactical environment "Digital Letters": Which of these scripts can paint the letter U on the stage?

before, after or inside the loop – see Figure 2, tasks a, d);

- repetition of more commands in a loop (the pupil creates the right order of Repeat commands – see Figure 2, tasks b, e);
- using scenarios with several subsequent loops (see Figure 2, task c);
- reading of a script with repetition (the pupil says what the script carries out);
- another didactical environment, some other activity is carried out (stamping pictures, changing costumes, drawing shapes from lines) with a similar use of the Repeat command in the script;
- problem task with experimenting, a more complex task (the pupil recognizes what may be repeated);
- suggesting solutions without testing them on the computer;
- modifying a script to solve a related task;
- correcting a wrong script to make it correspond to the described situation;
- script-writing optimization (the pupil shortens the script of the same activity – see Figure 8);
- distinguishing between commands that must be executed only once (e.g., *pen down*) and commands that must be repeated (*move forward*).

« 45 » If possible, the first activities introducing a given concept were set in didactical environments that were already familiar to the pupils. For example, chapter 3 – Procedures uses the environment "Train" from the introduction chapter on how to create a script. The same environment is also used in the penultimate chapter 9 – Procedures and parameters. Thus, pupils did not have to get to know a new environment and could focus fully on the newly introduced concept.

« 46 » Having solved a set of tasks up to a certain level of difficulty, the didactical environment changed. The sprites in Scratch carried out a different activity and the level of difficulty of the tasks decreased again. The intention was to show the general nature of the use of the given concept and also to allow weaker pupils to think about tasks at a level that was more acceptable for them. For example, in the chapter on Objects and Communication (broadcasting messages), tasks of the same difficulty are progressively set in environments in which sprites change costumes while dancing or making magic, or move or "talk" to one another. Each of the environments allowed for a creation of problems that showed the concept in a more complex situation, from a different point of view or in another activity.

« 47 » While preparing the curricula, I identified several types of pupils' activities. These were then reduced to the following four major types of pupils' activities and then used in methodological materials for teachers:

Discovery – collective activity. Pupils first try something on the computer, experiment, observe, inquire, and then, guided by the teacher report, talk to the others. An example of this type of activity in the didactical environment "Train" (Figure 2) could be: *Try to add a block* [pen up] *to a script drawing a train. Say what you see after the script is executed. Run this script once more – what has happened? Explain.*

Individual work – the pupils' task is to create some script; they work in pairs or individually at their own pace. Often, they work on a set of tasks that may follow on from one another. These activities may contain creative tasks or short projects. The teacher's role is to visit pupils or groups individually, to monitor, check, consult, encourage, advise, and assign the next task.

Discussion – collective activity, pupils think over a given situation or problem and try to interpret it. In some tasks they may look for a mistake in the script. The teacher plays the role of moderator, not the decisive authority. The goal is, among others, the development of pupils' ability to present arguments (Figure 9).

Reading the program – individual activity. Pupils read the script and without using a computer they say, draw or write what the script will carry out. If pupils do not reach a consensus when checking it together, the script can be re-written into a computer, which then becomes the decisive authority.

« 48 » These four major pupil activities are not related one-to-one to the stages of development pupils' responses to open-ended questions as in SOLO Taxonomy (Biggs & Collis 1982). In each type of activity, we can find activities related to more stages of SOLO, which, for example, require following, describing, comparing, explaining and analyzing. The types of activities used were related more to the knowledge and skills required by the curricula materials (NÚV 2018), e.g., to create a program to solve a problem, to read the code and explain what will happen, to find and correct a mistake in a program, to change the code so that it will solve a similar problem. For example, the *Discussion* activity is quite appropriate for fulfilling the response of reading and explaining the code, and in this type of activity you can find tasks requiring skills from both lower and higher stages of SOLO or Bloom's taxonomy.

Pilot evaluation study

« 49 » When creating materials based on the approach and principles described above, we came across questions of relevancy, consistency, practicality and effectiveness (Nieveen & Folmer 2013). One of the questions was whether teachers would be willing and able to accept this method, another was whether it would help pupils to achieve the goals. Where are the weaknesses? Where are the activities that deviate from the direction of building the given concept? Where is there an insufficient number of activities for practice or to support understanding? In this pilot study, we decided to use a heuristic approach.

« 50 » Piloting was conducted through design research that looked to develop an evidence-based curriculum intervention with three stages of piloting at schools. All the piloting was conducted within regular lessons. The method of participant non-structured observation according to Roman Švaříček and Klára Šeďová (2007) was used. Each stage was followed by the reworking or adjustment of educational materials based on analysis of the piloting. The frame (prepared in advance) was enriched by activities for 30 hours of teaching. A manual for teachers was also developed, in which the goal, main concept and methodological recommendations of each activity were described. Tips on how to conduct the lesson, correct solutions and troubleshooting were also included.

« 51 » The assumption was that teachers would not be familiar with the constructivist approach to education. Thus, the manual for teachers focused on how to give the lesson. Instructive teaching was not recommended, teachers were dissuaded from demonstrating to their pupils how to work in the environment or how to drag blocks. The materials emphasized the importance of supporting pupils in looking for solutions on their own, relying on their own activity, exploration and manipulation. We thought that inexperienced teachers needed the greatest support, so each activity was supplemented by an overview of possible mistakes and procedures for how to react to these mistakes quickly. For example, the disappearance of the sprite from the stage might be caused by several mistakes (hiding the sprite using a command, covering the sprite with another one, or because the sprite "runs away from the stage"). For an inexperienced teacher it is difficult to consider all of the possible causes quickly enough to deal with the situation.

« 52 » In the first stage, the new materials were piloted in lessons taught by experienced teachers in three ordinary classrooms. Each group consisted of 13 pupils aged 12–13, on average. One of the authors was present in each of the lessons and each lesson was followed by a period of analysis with the teacher. This way of creating and piloting enabled us to respond to pedagogical situations and problems that arose in the lessons and to modify the development of the subsequent chapters. It also allowed us to choose a more appropriate model of giving feedback in the next phase, in which we described hard situations in the classroom from the educational point of view, like "it works incorrectly and nobody knows why," and pupils' most frequent mistakes. At the end of this piloting, some activities were added and some were left out or reorganized. The manual for teachers was modified to ensure it was useful for teachers with no prior experience.

« 53 » Seven ordinary classrooms from seven different schools with pupils of the same age, both girls and boys, entered the second stage of piloting. The teachers involved in this stage of piloting had no knowledge of programming and no prior experience of teaching it. This lack of experience was partially compensated for by these teachers' not having developed any habits through teaching programming with traditional methods and thus being able to follow this innovative approach more naturally and easily. The teachers did not attend any training. This was because we wanted to verify that the developed curriculum was comprehensible for teachers in everyday school practice (where teachers are unlikely to get any special training and will have to rely on the activities and manuals only). We wanted to see if the materials could be used for teaching and what limitations were connected with them. There was one webinar giving instructions on how to orient oneself in the materials, how to use the teachers' manual and how to organize lessons.

« 54 » The teachers wrote about their experience after each lesson in the form of a blog: to what extent they had managed to follow the methodological instruction, what their pupils coped with, what the obstacles were, and what recommendations they would give. They could add a file with pupils' work or mistakes in order to better describe the situation. Some of the teachers occasionally used this tool to present some of the pupils' excellent work of which they were proud. The teachers were supervised by methodologists from five participating faculties of education, who visited them now and then and could help

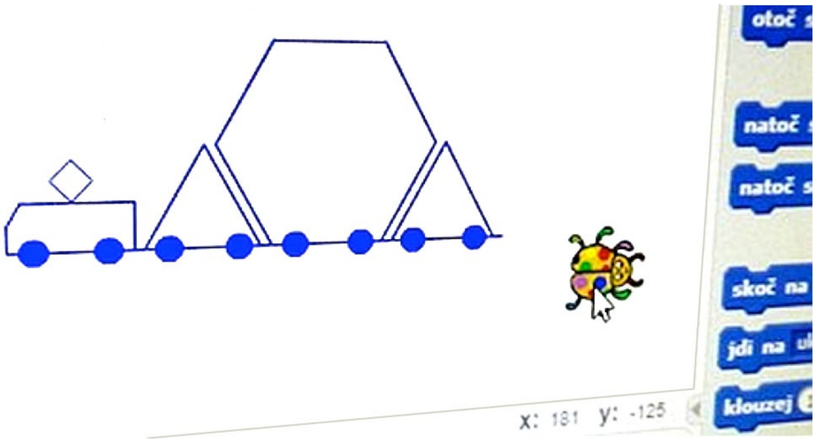


Figure 10 • A creative 12-year-old pupil's work; programming of his own train carriages as new procedures. Photographed by a teacher.

them. They also had the chance to contact the authors by email when some obstacles occurred, and authors responded to the reports if necessary. The analysis of teachers' logs was made in the team of authors by way of a collective case-study approach. The aim was primarily to find areas in the curricula that could be improved, typically when more respondents demonstrated too large a knowledge gap between consecutive tasks – thus revealing a lack of appropriate tasks for creating enough isolated models. This stage was concluded by further modification of the materials and the creation of a second version.

« 55 » The third stage of piloting, much like the second stage (with new inexperienced teachers, some of whom will take part in the newly created course of in-service teacher training), will be conducted this year. The curricular materials are expected to be finalized based on experience from the third piloting (Figure 10).

Findings and discussion

« 56 » I am aware of the limitations of the chosen research method; the observations of teachers offered authentic but subjective findings and were triangulated by authors' observations. However, I can present some of the findings from the piloting (which included both experienced teachers and teachers who lacked education in informatics and had no prior experience in teaching programming). I will discuss them shortly. So far, the piloting has shown that:

- Pupils needed more than 10 lessons, i.e., more than two months, to get to understand the way of working in a programming environment well enough to be able to work systematically and to look for their mistakes using an intentional procedure. Until then they were able to carry out individual tasks but often departed from the already mastered procedures of creating a script. This took the form of, for example, deleting all badly done script and starting again from the very beginning, or by random exchanging of individual blocks in the script by the trial-and-error method (without any system that would aim to make the script functional). Even if pupils mastered a more sophisticated universal procedure of correcting the code, they often dropped it as if they did not believe in it or had forgotten it.
- The development of general skills such as creating a script or reading and correcting it was not linear, but happened in waves. Sometimes, having met a new didactical environment, pupils returned to lower levels of understanding. It can be said that the creation of a generic model of how to edit script took longer than, for example, grasping the concept of repetition.
- Topics where mathematics and particularly geometrical knowledge was used or required (non-right-angle rotation, counting to a whole or straight angle, coordinates) were perceived by pupils as very difficult.
- Even experienced teachers found it difficult to adapt to a style of teaching in which they were not expected to talk too much to their pupils but let them work independently.
- Teachers needed several lessons, often for more than a month, to find a way of organizing their teaching and of applying the methodological recommendations for pupils' active involvement. Meanwhile, some of them perceived their lessons as disorganized. After managing the teaching style and getting familiar with the approach, they showed more self-efficacy and focused more on the pupils, their progress, difficulties and mistakes.
- Teachers found it difficult to discover the sources of mistakes in their pupils' scripts. They had no experience with this and thus did not feel confident enough.
- Some teachers skipped those tasks that they found too similar to previous tasks because they did not see that these tasks allowed the pupils to build other isolated mental models of the learned concept.
- Teachers tended to evaluate each pupil's answer and state whether it was right or wrong, instead of allowing the computer to provide the feedback on whether the pupil's idea was correct or not. Teachers have to learn to say "let's try it in Scratch," which means "create a script for your idea, launch it and look."
- If pupils asked for help when they made a mistake, teachers immediately showed them the mistake instead of giving encouragement or advice on how to look for the mistake.
- But teachers found pupils' attitudes to programming positive, which was motivating for them as teachers.

JIŘÍ VANÍČEK
is Associate Professor and Head of the Department of Informatics at the Faculty of Education, University of South Bohemia in České Budějovice, Czech Republic. He takes care of informatics teaching in elementary and high schools, early-age programming education and computer-assisted learning of mathematics. He has published over 60 scientific papers and over 50 methodological works. He has written six information technology and programming textbooks for elementary and junior high schools. He is a part of an expert group for the innovation of national informatics curricula at the National Institute for Education. Since 2017 he has been the head of the strategic ministerial project PRIM – Support for the Development of Computational Thinking. The main goal of this project is to develop new informatics curricula for all levels of schools (including elementary and kindergarten) and to create a system for the education of teachers in this field. He is country representative of the Czech Republic in the International Bebras Committee. For 11 years he has been organizing the Bebras informatics contest in the Czech Republic.

« 57 » We can distinguish two main groups of obstacles mentioned by teachers and observers. On the one hand, obstacles that arose from the organization of lessons and using new methods of work with pupils, and, on the other hand, obstacles caused by insufficient programming knowledge and experience of the teachers. Most of the teachers were affected by the first type of difficulties, and this was visible even in teachers who had taught programming before and already had a particular teaching style. The second type of obstacles, for example problems with finding mistakes or understanding more complex tasks, occurred in teachers with a low level of programming knowledge.

« 58 » I consider the first, pedagogical type of obstacles, as more difficult to solve because teachers could have a tendency to not follow the required approach and to return to the previous way of teaching. It is a premise that if these teachers teach once more and obtain some experience, difficulties of the second, more expert approach, will recede. They probably would choose the tasks they understand better and also could gain some self-confidence.

« 59 » The pilot study could not answer the question of whether the described approach to teaching programing is more effective or whether pupils gain better outcomes. We could begin to answer those questions by the mass expansion of the method and with experienced teachers. There will always be a problem with finding a control research group taught traditionally, because programming, for this age group of pupils, has not yet been taught on a large scale.

Conclusion

« 60 » In the article, the principles of creating early programming curricula based on concept building have been presented. The theory taken from didactics of mathematics was applied to programming education by reworking the learning aims, using didactical programming environments, using sequences of short tasks targeted at one chosen concept, and using teaching methods that supported pupils' activity. Based on these principles, educational materials were developed (Vaníček, Nagyová & Tomcsányiová 2019). Then the research design of evaluating these materials, which is conducted in three stages, was described and findings were presented that are helpful for the adaptation of these materials to ensure that teachers with a lack of experience and programming skills will be comfortable using them.

« 61 » Learning programming as a playing field for the development of computational thinking can contribute significantly to the development of each pupil. To achieve this goal, it is important that pupils do not simply adopt ready-made knowledge but learn to argue, discuss and evaluate. Critical thinking, as a way to work out what is disinformation, is an important skill for citizens of a democratic society to learn. In our geographical area, in particular, school should put an emphasis on educating pupils in not allowing anybody to manipulate them. This will lead to the fulfillment of the thesis of the well-known educator Comenius, who believed education would help to solve the problems of the whole world.

Appendix

Online textbooks for programming in Scratch referred to in Figure 7.

- Brennan K., Hawson J. & Chung M. (2012) Scratch curriculum guide. http://scratched.gse.harvard.edu/resources/scratch-curriculum-guide
- Chiang J. (2012) Scratch lessons: Shall we learn scratch programming for tweens. http://scratched.gse.harvard.edu/resources/scratch-lessons-shall-we-learn-scratch-programming-tweens
- Hawk D. (2012) Learn Scratch. https://mywebspace.wisc.edu/dhawk/scratch/
- Nováková V. (2013) Možnosti využití jazyka Scratch na ZŠ [Potentials and ways how to use Scratch at basic school education] Bachelor thesis, Faculty of Education, University of Karlova, Czech Republic. https://is.cuni.cz/webapps/zzp/detail/75553/?lang=en
- Wilson A. (2012) Introduction to Programming. http://scratched.media.mit.edu/resources/introduction-programming

Acknowledgements

I would like to thank the project "PRIM" (Support for Development of Computational Thinking) – CZ.02.3.68/0.0/0.0/16_036/0005322 – for funding this work.

References

Ackermann E. (2001) Piaget's constructivism, Papert's constructionism: What's the difference? Future of Learning Group Publication 5(3): 438. http://learning.media.mit.edu/content/publications/EA.Piaget%20_%20Papert.pdf

Ackermann E. (2010) Constructivism(s): Shared roots, crossed paths, multiple legacies. In: Clayson J. E. & Kalaš I. (eds.) Constructionist approaches to creative learning, thinking and education: Lessons for the 21st century. Proceedings of the Constructionism 2010 Conference. American University of Paris, Paris: 1–9. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.473.6201&rep=rep1&type=pdf

Benton L., Hoyles C., Noss R. & Kalas I. (2016) Building mathematical knowledge with programming: Insights from the ScratchMaths project. In: Proceedings of Constructionism 2016. Suksapattana Foundation: Thung Khru, Thailand: 25–32.

Biggs J. & Collis K. (1982) Evaluating the quality of learning: The SOLO Taxonomy. Academic Press, New York.

Bruner J. S. (1960) The process of education. Harvard University Press, Cambridge.

Dagienė V., Sentance S. & Stupurienė G. (2017) Developing a Two-Dimensional Categorization System for Educational Tasks in Informatics. Informatica 28(1): 23–44.

Dubinsky E. & McDonald M. (2001) APOS: A constructivist theory of learning in undergraduate mathematics education research. In: D. Holton (ed.) The teaching and learning of mathematics at university level: An ICMI study. Kluwer, Dordrecht: 275–282.

Gander W. (2014) Informatics and general education. In: Gülbahar Y. & Erinç K. (eds.) Informatics in schools, teaching and learning perspectives. Springer, Heidelberg: 1–7.

Gray E. & Tall D. (1994) Duality, ambiguity and flexibility: A proceptual view of simple arithmetic. Journal for Research in Mathematics Education 25(2): 116–141.

Grover S., Pea R. & Cooper S. (2015) Designing for deeper learning in a blended computer science course for middle school students. Computer Science Education 25(2): 199–237.

Hansen A. K., Hansen E. R., Dwyer H. A., Harlow D. B. & Franklin D. (2016) Differentiating for diversity: Using universal design for learning in elementary computer science education. In: Proceedings of the 47th ACM Technical Symposium on Computing Science Education. ACM, Memphis, Tenessee: 376–381.

Hejný M. (1987) Teória vyučovania matematiky 2 [Theory of mathematics education 2]. Slovenské pedagogické nakladateľstvo, Bratislava.

Hejný M. (2004) Mechanizmus poznávacího procesu [Mechanism of cognitive process]. In: Hejný M., Novotná J. & Stehlíková N. (eds.) Dvacet pět kapitol z didaktiky matematiky [25 chapters of didactics of mathematics]. PedF UK, Praha: 23–42.

Hejný M. (2012) Exploring the cognitive dimension of teaching mathematics through a scheme-oriented approach to education. Orbis scholae 6(2): 41–55.

Hejný M., Slezáková J. & Jirotková D. (2013) Understanding equations in schema-oriented education. Procedia – Social and Behavioral Sciences 93: 995–999.

Kalaš I. (2010) Recognizing the potential of ICT in early childhood education. UNESCO IITE, Moscow.

Kline M. (1973) Why Johnny can't add: The failure of the New Mathematics. St. Martin's Press, New York.

Krejsa J. (2014) Výuka základů programování v prostředí Scratch [Education of basic programming in Scratch environment]. Master's Thesis, Faculty of Education, University of South Bohemia in České Budějovice, Czech Republic.

Meerbaum-Salant O., Armoni M. & Ben-Ari M. (2013) Learning computer science concepts with Scratch. Computer Science Education 23(3): 239–264.

Nieveen N. & Folmer E. (2013) Formative evaluation in educational design research. In: Plomp T. & Nieveen N. (eds.) Educational design research. Netherlands Institute for Curriculum Development (SLO), Enschede: 152–169.

NÚV [National Institute for Education] (2018) Návrh revizí rámcových vzdělávacích programů v oblasti informatiky a informačních a komunikačních technologií [Proposal for the revision of the national curriculum in the field of informatics and ICT]. NÚV, Praha. http://www.nuv.cz/file/3362/

Piaget J. (1951) The psychology of intelligence. Routledge, London. French original published in 1947.

Piaget J. (1952) The origins of intelligence in children. International University Press, New York. French original published in 1936.

Resnick M. (2017) Lifelong kindergarten: Cultivating creativity through projects, passion, peers, and play. MIT Press, Cambridge MA.

Resnick M., Maloney J., Monroy Hernández A., Rusk N., Eastmond E., Brennan K., Millner A., Rosenbaum E., Silver J., Silverman B. & Kafai Y. (2009) Scratch: Programming for all. Communications of the ACM 52: 60–67.

Selby C. C. & Woollard J. (2013) Computational thinking: The developing definition. Project report, University of Southampton. https://eprints.soton.ac.uk/356481

Švaříček R. & Šeďová K. (2007) Kvalitativní výzkum v pedagogických vědách [Qualitative research in pedagogical science]. Portál, Praha.

Tall D. & Vinner S. (1981) Concept image and concept definition in mathematics with particular reference to limits and continuity. Educational Studies in Mathematics 12: 151–169.

The LEAD Project (2014) Super scratch programming adventure! (Covers version 2): Learn to program by making cool games. No Starch Press, San Francisco. Chinese original published in 2010.

Vaníček J. (2015) Programming in Scratch using inquiry-based approach. In: Brodnik A. (ed.) Informatics in schools: Curricula, competencies, and competitions. Springer, Heidelberg: 82–93.

Vaníček J., Nagyová I. & Tomcsányiová M. (2019) Programování ve Scratch pro 2. stupeň základní školy [Programming in Scratch for lower secondary schools]. Textbook, beta version. https://imysleni.cz/ucebnice/programovani-ve-scratchi-pro-2-stupen-zakladni-skoly

Wing J. (2011) Research notebook: Computational thinking – What and why? The Link: The Magazine of the Carnegie Mellon University School of Computer Science 6: 20–23. https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

Wittmann E. C. (2001) Developing mathematics education in a systemic process. Educational Studies in Mathematics 48(1): 1–20.

Received: 16 January 2019
Accepted: 22 March 2019

Open Peer Commentaries

on Jirí Vanícek's "Early Programming Education Based on Concept Building"

Teaching Basic Concepts of Programming to All Children

Ágnes Erdősné Németh
Eötvös Loránd University, Hungary
erdosne/at/blg.hu

> **Abstract** • Vaníček's target article presents a pilot study of an introductory course of programming. In my commentary I argue that teachers will be a key factor of success if the pilot program is implemented in a top-down way. Other important issue are the differentiation between children with different abilities while they go through the tasks, and the continuation of learning programming after the entry-level course.

Familiarization with computational thinking is the first step in learning programming

« 1 » In 2013, the report "Informatics education: Europe cannot afford to miss the boat" of the joint Informatics Europe & ACM Europe Working Group on Informatics Education chaired by Walter Gander suggested: "Digital literacy should indeed be a required part of the education of all Europeans." It claimed that digital literacy was not enough, and that informatics and programming should be learned by all children by the age of 12 at the latest. To satisfy the formulated recommendations, there are two possible solutions:

- Bottom-up: In the UK, IT professionals and teachers joined together and founded the organization "Computing in schools." They organized themselves into networks, supported and taught one another, bringing IT education into schools at all levels.
- Top-down: As reported by Maciej Sysło and Anna Beata Kwiatkowska (2015), in Poland a new curriculum for teaching computational thinking and programming was prepared by experts. The ministry of education made this curriculum obligatory in all schools.

« 2 » Jeanette Wing (2011) argued that computational thinking should be taught to all children in the 21st century. An increasingly widespread model for concept-based learning of computational thinking is the Bebras model. It is a very successful model to "wrap up serious scientific problems of informatics and the basic concepts into playful tasks, inventive questions thus attracting students' attention" (Dagienė & Stupurienė 2016: 26).

« 3 » In his target article, Jíří Vaníček describes a pilot study on how a concept-based learning model can be used in teaching programming to children at the age of 12. It seems that all participants in the pilot study have already met computational thinking basics at Bebras competitions. The described learning model is planned to be introduced in schools as part of the compulsory curriculum, in a top-down way, in 2021. The way developed in the pilot study seems to be a good initiative to implement the above recommendations for all students at once, through a well-designed, tried and tested system. The reason for this is that it relies on a concept-based learning model and a problem-type-oriented method, as well as incorporating basic computer science concepts in a logical order as it written in §34.

Problem type-oriented method of teaching programming

« 4 » According to Péter Szlávi and László Zsakó (2003), the most effective method of teaching programming to a wide variety of students – especially in primary and secondary education – is the problem-type-oriented method. "The basic idea of this method is to explain new programming terms and methods through expanding the programming problems" (Bernáth & Zsakó 2017: 41). And the "effectiveness of the […] method depends largely on the tasks we choose and the strategies we apply […]" (ibid).

« 5 » In Vaníček's article a problem-type-oriented method is used to teach the basic elements of programming from the task-oriented point of view, on the base of concepts. The described tasks are built on top of one another, new terms are introduced in a task and the new knowledge is used immediately in the next task(s). The series of tasks take advantage of applying the new knowledge, because using something builds and tests a higher level of understanding.

« 6 » Vaníček follows the constructionist approach by involving the teacher as a mentor in the whole process and by leaving the children to learn at their own pace and according to their own understanding.

The importance of teachers

« 7 » I agree with the author in §4 about the importance of the experience and training of the teacher, and their ability and willingness to try a teaching model involving more creative activities.

« 8 » However, this raises several questions: Will teachers agree to teach programming in mathematics classes? How will they manage to become familiar with their new role in this teaching method? Will they cope well with stepping outside their comfort zones both didactically and professionally? So, I wonder how future research into changing teachers' attitude could find answers to these questions. (Q1)

Scratch! What is next?

« 9 » There are opposing opinions about whether text-based or block-based programming is better to begin with. David Weintrop and Uri Wilensky (2015) wrote that block-based programming is easier to get started with, and it is more engaging for the learners at first. But Colleen Lewis reported that she "[…] found that students that learned Logo had on average higher confidence in their ability to program" Lewis (2010: 346).

« 10 » Over the last few years, I have researched how to teach children programming. In my PhD thesis (Erdősné 2019), I describe a system from upper-primary to the end of secondary school, from teaching LOGO at first for all, through the problem-type-oriented way, to becoming proficient. I found that a long-term engagement in programming requires entry-level courses to be followed by further courses. In the short pilot program described in the target article, children learn the basic concepts of programming through interesting tasks in a concept-based way. While getting this course included in the curriculum in a top-down way is a step forward, it would be interesting to see plans for the continuation of the course for children who want to know more and who might even want to become computer scientists.

« 11 » Given these still unsolved controversies, why was Scratch the chosen language in the pilot study reported in the target article, and which languages will be taught after the entry-level course? (Q2)

Talented students and those making slower progress

« 12 » In the pilot study, there were only small groups. In these groups the children could and wanted to proceed together, as they had reasonably similar abilities. In order to scale up this method – to a whole educational system with huge differences in ability among children in larger groups and in different schools – it would be crucial to include tasks for the children outside the mainstream: specifically, more complicated tasks for the talented and practice tasks for those falling behind.

« 13 » So, it could prove useful to extend the task repository with new ideas and to give some tasks outside the main focus for faster students, to let them spend their time creatively and usefully while waiting for the slower ones to understand the concept being practised. For example, Paula Bonta, Artemis Papert and Brian Silverman (2010) offer a good idea, which is to have faster students draw TurtleArt pictures within the Scratch environment.

References

Bernáth P. & Zsakó L. (2017) Methods of teaching programming – strategy. In: Stoffová V. & Horváth R. (eds.) Proceedings of the international scientific and professional conference XXXth DIDMATTECH. Part 1. Trnava University, Trnava: 40–51. http://didmattech.truni.sk/2017/proceedings.pdf

Bonta P., Papert A. & Silverman B. (2010) Turtle, art, TurtleArt. In: Clayson J. E. & Kalaš I. (eds.) Constructionist approaches to creative learning, thinking and education: Lessons for the 21st century. Proceedings of the Constructionism 2010 Conference. American University of Paris, Paris: 1–9.

Dagienė V. & Stupurienė G. (2016) Bebras – A sustainable community building model for the concept based learning of informatics and computational thinking. Informatics in Education 15(1): 25–44. https://files.eric.ed.gov/fulltext/EJ1097494.pdf

Erdősné Németh Á. (2019) From LOGO till Olympiads: Talent management in grammar school. Unpublished PhD thesis. ELTE IK Doctoral School, Hungary. https://edit.elte.hu/xmlui/handle/10831/41609

Lewis C. M. (2010) How programming environment shapes perception, learning and goals: Logo vs. Scratch. In: Lewandowski G., Wolfman S., Cortina T. J., Walker E. L. & Musicant D. R. (eds.) Proceedings of the 41st ACM technical symposium on Computer Science Education (SIGCSE '10). ACM, New York: 346–350.

Sysło M. M. & Kwiatkowska A. B. (2015) Introducing a new computer science curriculum for all school levels in Poland. In: Brodnik A. & Vahrenhold J. (eds.) Informatics in schools: Curricula, competencies, and Competitions. Springer, Heidelberg: 141–154.

Szlávi P. & Zsakó L. (2003) Methods of teaching programming. Teaching Mathematics and Computer Science 1(2): 247–257.

Weintrop D. & Wilensky U. (2015) To block or not to block, that is the question: Students' perceptions of blocks-based programming. In: Proceedings of the 14th international conference on Interaction Design and Children (IDC '15). ACM, New York: 199–208.

Wing J. M. (2011) Research notebook: Computational thinking – What and why? The Link: The Magazine of the Carnegie Mellon University School of Computer Science 6: 20–23. https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

Ágnes Erdősné Németh teaches mathematics and informatics at Batthyány High School in Nagykanizsa, Hungary. Many of her students have made it to the final rounds of national programming competitions, including the Central European Olympiad in Informatics and International Olympiad in Informatics. She is a vice-president of the John von Neumann Computer Society, where she is responsible for national programming contests. She has authored several papers about the methodology and didactics of teaching informatics. Her current research interest is teaching computational thinking for all and teaching computer science for talented pupils in upper-primary and secondary school.

Received: 31 May 2019
Accepted: 16 June 2019

Programming or Problem Solving with Computers?

Maciej M. Sysło
Warsaw School of Computer Science, Poland
syslo/at/ii.uni.wroc.pl

> Abstract • I put the proposed approach to learning programming against the place of programming in the curriculum of computer science, e.g., as introduced in Poland. Then I comment on some drawbacks of focusing mainly on programming when developing concepts that belong to much wider areas.

« 1 » The commentary is heavily based on my experience in developing computer science (CS) education in Poland over the last 30 years. In the new CS curriculum (Sysło & Kwiatkowska 2015), five knowledge areas in the form of general requirements (as "Unified aims"), are the same for all school levels. The content of each aim (called "Attainment targets") is defined adequately according to the school level. The benefits of such a spiral curriculum are presented by Mary Webb et al. (2017: Section 5). The most important aims, from a CS-education point of view are the first two aims and their order in the curricula: (a) *Understanding and analysis* of problems based on logical and abstract thinking, algorithmic thinking, and information representations, and then (b) *Programming and problem solving by using computers* and other digital devices – designing algorithms and programs, organizing, searching and sharing information, using computer applications. Consequently, programming is a tool in the process of comprehending and developing CS concepts and computational thinking when solving problems in various areas and school subjects. Our team's motto is: "First think computationally then program."

« 2 » The approach to teaching and learning CS proposed by Jiří Vaníček in his target article is just the opposite. The concepts listed in Figure 7 refer to programming and the curricular materials illustrated in Figures 2–5 are also mainly used to develop the understanding of concepts within a programming environment. However, a programming language is only a tool used in the final stage of the process of solving a problem. Students may also learn CS concepts in an unplugged approach or using ready-to-go programming environments such as The Hour of Code (http://hourofcode.com and http://code.org). The main goal of teaching CS is to equip students with problem-solving skills, but this is not restricted to CS only. Therefore, students start by analysing a given problem situation that uses meaningful objects, discovering concepts and heuristic methods for solving the problem. Then they think computationally about the objects and concepts, they discover algorithms and come up with a solution. Finally, they may program, test, and debug the solution in a visual/block- or text-based programming environment.

« 3 » Programming is a tool in the process of developing concepts and computational thinking when approaching and solving problems in various areas and school subjects. *The purpose of programming is performing abstraction not writing programs* since every program is an effect of mainly abstract thinking on a problem situation to be solved.

« 4 » CS is much more than programming. You are going to propose "practical curricula of programming" (§24) and write textbooks for the school subject "programming." Is this school subject to be independent in the curriculum or to be a part of the CS curriculum? (Q1) In my opinion, as justified in §§1–3 above, the former choice is not the right direction for teaching programming.

« 5 » I do not agree with Vanicek's statement that "[beaver tasks] are hard to solve without an understanding of the concepts related to programming" (§11). While the results have not been published yet, from the data we have been collecting so far it is clear that, while most of our Polish participants in the Bebras contest are not familiar with programming, some of them get the highest possible score.

« 6 » In the sense described above in §§1–3, learning programming concepts should be independent of a programming language and its environment. A programming language should be used mainly for communication with computers about algorithms using programming constructions that are similar in various programming languages. So, regarding the textbooks mentioned in §14, in particular Vaníček, Nagyová & Tomcsányiová (2019), I would suggest an approach that can be described by the label "Programming *and* Scratch" rather than "Programming *in* Scratch."

« 7 » In §19 Vaníček states that one of the disadvantages of block-oriented programming environments like Scratch is that students can see most of the blocks of the language while they do not master all of the commands yet. To remedy this problem, I can only suggest using the environments of The Hour of Code and code.org. They contain courses of progressive and spiral learning that successively add blocks with increasingly advanced commands. These two environments may play a preliminary role in learning how to program before going to environments of programming languages where students have to build their programs from scratch. Let me add that these two environments are very popular in Poland. Recent statistics from the organizers of these initiatives show that, in this regard, students and teachers from Polish schools are among the five top countries in the world.

« 8 » I am surprised by §53: "The teachers involved in this stage of piloting had no knowledge of programming and no prior experience of teaching it." I cannot even imagine how one can convince a teacher to teach an unknown subject! From the pedagogical point of view, learning how to program (for a teacher) is not the same as learning how to teach programming to students. Moreover, programming is a skill that demands practice, similarly to using any natural language.

« 9 » Finally, although the approach presented in the target article is based on "characteristics of effective mathematics teaching" (§22) one has to take into consideration that the two subjects, mathematics and CS (especially programming), differ substantially on several issues. The most important issues are:

- Mathematics as it is taught in schools is a "theoretical" subject in the sense that applications do not drive its teaching, whereas CS in many respects is an experimental and widely applicable subject, even in schools;
- Teachers of mathematics are usually graduates in mathematics from universities, whereas, apart from a small percentage of CS teachers, most come to teach CS as a second subject with no solid prior knowledge of CS.

« 10 » There are many ways to apply a constructivist approach to CS education, especially in teaching how to program. However, programming is only a tool and to propose a right place for it is crucial for the success of CS education. Although it is based on some constructivist ideas, the proposal of teaching programming presented in the target article does not appear to be the right choice.

References

Sysło M. M. & Kwiatkowska A. B. (2015) Introducing a new computer science curriculum for all school levels in Poland. In: Brodnik A. & Vahrenhold J. (eds.) Informatics in schools: Curricula, competencies, and Competitions. Springer, Heidelberg: 141–154.

Vaníček J., Nagyová I. & Tomcsányiová M. (2019) Programování ve Scratch pro 2. stupeň základní školy [Programming in Scratch for lower secondary schools]. Textbook, beta version. https://imysleni.cz/ucebnice/programovani-ve-scratchi-pro-2-stupen-zakladni-skoly

Webb M., Bell T., Davis N., Katz Y. J., Reynolds N., Chambers D. P., Sysło M. M., Fluck A., Cox M., Angelivalanides C., Malynsmith J., Voogt J., Zagami J., Micheuz P., Chtouki Y. & Mori N. (2017) Computer science in the school curriculum: Issues and challenges. In: Tatnall A. & Webb M. (eds.) Tomorrow's learning: Involving everyone. Learning with and about technologies and computing. WCCE 2017. IFIP Advances in Information and Communication Technology 515. Springer, Cham: 421–431.

Maciej M. Sysło is a professor of mathematics and computer science, an academic and school teacher, an instructor at in-service courses for teachers, organizer of the Bebras Competition, author of computer science curricula, educational software, school textbooks and guidebooks for teachers, a member of several national and international committees on education, a Polish representative to IFIP T3, and a member of the Council for Informatization of Education at the Ministry of National Education. As a leading author, he proposed a new computer science curriculum for all school levels (K-12). The curriculum is unified according to the five "Unified aims" of learning computing, including programming through 12 years of school education. It was introduced to primary schools (1–8) in 2017 and will be introduced to high schools in 2019.

Received: 4 June 2019
Accepted: 30 June 2019

Expected Constructivist Teaching of Programming: Necessity and Computational Perspective

Chantal Buteau
Brock University, Canada
cbuteau/at/brocku.ca

> Abstract • Vaníček proposes a developed curriculum of basic programming informed by sound constructivist-based principles. Two questions for reflection are brought forth, one concerning the expected constructivist approach to teaching and one about the development of pupils' computational perspectives.

« 1 » As I am not a scholar of constructivism, I hesitated to agree to writing an open peer commentary on Jiří Vaníček's target article. But as I read through the article, I came across the following excerpt: "The question is how to put into action practical curricula of programming based on a methodology taken from the didactics of mathematics" (§24). Since I have taught an introductory programming-based university course for mathematics investigations and applications offered at my institution for over 15 years, and have examined that kind of teaching and learning for a decade (e.g., see the target article by Buteau, Sacristán & Muller 2019, this issue), it is from the perspective of a constructivist practitioner of teaching programming for mathematics that I provide two questions in what follows in the hope of bringing some interesting points to reflect upon for both Vaníček and the reader.

Intended "constructivist teaching" … but is it necessary?

« 2 » In §51 Vaníček highlights a position taken when developing a curriculum, and related materials, of programming for junior-high school: it was assumed that teachers would not have been familiar with the constructivist approach to education, and that as such, teacher manuals explicitly addressed how to give the lessons.

« 3 » It made me wonder what arguments were brought up in the manuals to convince teachers to adhere to this different teaching approach. After all, they would receive all material and, unless convinced otherwise, could easily decide to use the material with their usual teaching approach. I reflected back on my own experience. When I first taught the introductory programming-based course for mathematics mentioned earlier in §1, I was provided with all the material. I was also provided with significant guidance from one of my colleagues, as I taught my course section alongside him, using (what I now know to describe as) a constructivist approach.

« 4 » And now, I could hardly imagine teaching the course with this same material yet using a more traditional approach. As such, I initially took for granted what seems to be Vaníček's position. But it is not very clear, is it? All teachers involved in the first two stages of Vaníček's piloting were observed in the classroom, and as such, we could assume, might have been more inclined to closely follow the teacher manual in which the constructivist approach to teaching is strongly recommended. It seems natural to suspect that some teachers, when not observed in their classrooms, may decide to continue with their usual approach to teaching. My practical question is: When using the curricular material developed based on constructivist principles, to what extent is a constructivist approach to teaching necessary for a "successful" learning of programming by pupils? Or put differently: could this material be such that it would enrich pupils' learning of programming even when taught with a more traditional approach? And how relevant is this question, or should it be, for the curriculum developers? (Q1)

« 5 » In the future, it would be interesting to know whether some of the teachers "became more constructivist" in their teaching of other areas as they gained experience teaching the material developed for programming with a constructivist approach. This seems to be anticipated by Vaníček (see "Implications" in his abstract).

Teaching to develop pupils' computational perspectives? Relevant?

« 6 » Overall, it is with great interest that I read Vaníček's target article. It appears to be a sound approach to developing curricular material that will soon be needed in schools in his country as a response to a newly revised computing curriculum. The material developed does not, interestingly, assume any programming knowledge by teachers. In §33 the author illustrates how their approach led to a sequence of concept development similar, to some extent, to other approaches. As I was reading, in §26, about the principles grounding the approach taken I could not help but reflect on the three-dimensional framework of computational thinking development proposed by Karen Brennan and Mitchel Resnick (2012) that is summarized as follows:

"*computational concepts* (the concepts designers engage with as they program, such as iteration, parallelism, etc.), *computational practices* (the practices designers develop as they engage with the concepts, such as debugging projects or remixing others' work), and *computational perspectives* (the perspectives designers form about the world around them and about themselves)." (Brennan & Resnick 2012: 1)

« 7 » I could see, for example, that the principle mentioned in the 4th bullet point in §26 was associated with the computational-practice dimension. Earlier on, in §11, the author, crediting the work of Valentina Dagienė and colleagues (2017), acknowledges that the Bebras challenge contest involves "situational informatics tasks that simulate everyday life situations and their informatic dimension" (§11), which immediately made me think of the computational-perspective dimension. But, based on the description and examples provided in the target article, these kinds of tasks do not seem to have been integrated into the developed curriculum and material – or was it? I am left wondering about the author's position on the importance of exploiting the latter dimension in a curriculum of basic programming, even so since, as I reflected on my own practice as a teacher of the course mentioned earlier in §1, this dimension is unquestionably part of my teaching. My second question is: Which of the principles listed in §26 inform or could inform the development of tasks or sequences of tasks aiming at contributing to pupils' development of the perspective dimension of their computational thinking? Are there any specific aspects of a constructivist approach to teaching that could be key to this development? And to what extent is this question relevant to a curriculum developer? (Q2)

References

Buteau C., Sacristán A. I. & Muller E. (2019) Roles and demands for constructionist teaching of computational thinking in university mathematics. Constructivist Foundations 14(3): 294–309 (this issue). ▸ https://constructivist.info/14/3/294

Brennan K. & Resnick M. (2012) New frameworks for studying and assessing the development of computational thinking. In: Proceedings of the 2012 annual meeting of the American Educational Research Association. Volume 1. AREA, Washington DC: 25. http://web.media.mit.edu/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf

Dagienė V., Sentence S. & Stupurienė G. (2017) Developing a two-dimensional categorization system for educational tasks in informatics. Informatica 28(1): 23–44.

Chantal Buteau is Professor of Mathematics at Brock University (Canada). Since she joined Brock in 2004, she has been teaching an introductory programming-based course for mathematics investigations and applications to mathematics majors and prospective mathematics teachers. Her research interests in education mainly focus on the integration of digital technology for (university) mathematics learning, including programming, CAS, and epistemic mathematics computer games. Buteau currently leads a research project examining students' appropriation (i.e., instrumentalization) of programming for authentic mathematical exploration, simulation, and applications.

Received: 2 June 2019
Accepted: 7 June 2019

Learning Environments with Scratch

Michael Weigend
Holzkamp Gesamtschule,
Witten, Germany
mw/at/creative-informatics.de

> Abstract • The Scratch environment can be used to professionally design tasks for computer science education in schools. Working on tasks touching different facets of one and the same programming concept can help pupils to build up universal mental models. A problem is how to make this type of learning experience relevant for young people.

Curriculum design and relevance

« 1 » In §1 and §2 Jiří Vaníček points out the educational value of programming and computational thinking. However, computer-science topics listed in curricula represent the teachers' perspective: "To be well-educated citizens in a computing-intensive world and to be prepared for careers in the 21st century, our students must have a clear understanding of the principles and practices of computer science" (CSTA). But these "principles and practices" – as such – are not necessarily interesting for high-school students. The challenge for curriculum designers, teachers and textbook authors is to connect useful computer-science concepts with topics that are relevant for the younger generation.

Mental models and intuitions

« 2 » Vaníček suggests looking at the rich experience of maths education in order to get ideas for teaching programming. In §9 and §10, he presents Milan Hejný's theory of "generic models." According to this, students build up (mathematical) knowledge by constructing isolated models for given contexts first, which they use and test in different situations, thus developing more universal (decontextualized) "generic models." Vaníček advocates adopting this for informatics education. Additional to the theories mentioned in the article, one might consider "conceptual metaphors" that are used as "learning vehicles" in mathematics education (Lakoff & Núñez 1997). A conceptual metaphor is the transfer of (intuitive) knowledge from a well-known domain to a (still) unfamiliar domain. For example, the (familiar) concept of "motion along a line" is commonly deployed for understanding (unfamiliar) arithmetic. The addition of a (positive) number is represented by moving to the right for a certain number of steps while subtraction is moving to the left. Metaphors are used in informatics too. For example, terms like "loop" and "branch" are metaphors mapping physical motion to program execution.

Practicing versus discovering

« 3 » The constructionist approach is task-oriented: Create a relevant artefact and learn programming concepts and (accidently) discover "big ideas" on the way (Papert 2000). However, it is not easy to create a meaningful digital artefact from scratch. In §16 it is pointed out that teaching through larger projects like games or stories often requires organizing the process in detail ("step by step") through a tutorial. Indeed, the Scratch website offers some multimedia tutorials for animations and games. In the opinion of the author, following a tutorial is not the best way to gain deep knowledge of programming concepts, since this "does not help a pupil learn to generalize" (§19).

« 4 » Vaníček outlines a curriculum for an introductory informatics class that leads up to computational thinking. Its main characteristics (§26): There are many different short tasks focusing on one and the same concept ("etudes"). The students can check their result by themselves. The activities aim at various programming competencies (creating, reading, debugging, etc.). It embraces making mistakes and learning from them. In contrast to a project-oriented approach (with tutorials) the concept-oriented approach of the suggested curriculum could be called "bottom-up," since the students practice individual programming techniques and underlying concepts first, then use the acquired competence to create their own projects.

The educational power of Scratch

« 5 » The Scratch programming environment supports the constructionist idea. It has been designed with the intention "to nurture the development of a new generation of creative, systematic thinkers who are comfortable using programming to express their idea" (Resnick et al. 2009: 60). Again: It is not just about programming, it is about creating ideas on relevant topics, implementing and sharing them. Motivation is essential. The relevance of artefacts can be increased by embedding them in a social context: The young authors can publish their artefacts on the Scratch website in studios, they can allow commenting, observe the numbers of watches and "likes," etc.

« 6 » One way to learn programming – supported by the Scratch online platform – is to import a project and remix it to create something new. A basic problem of this approach is that it is often difficult to understand a Scratch program, which has been created by someone else. As Yasmin Kafai and colleagues (2009) observe, the technical quality of most programs made by young people in computer club houses in their leisure time is rather low.

« 7 » The article describes how to use the Scratch technology to design concept-oriented tasks ("etudes"). Vaníček gives examples of classroom activities (individual tasks, classroom discussions, etc.) based on Scratch-based "learning environments" that have been designed by teachers. Such a learning environment might consist of a series of tasks with increasing difficulty and a Scratch project with additional blocks (using "Make a Block") that can be used to solve these tasks. The additional blocks – if cleverly designed – make it possible to create extra short tasks that are easy to comprehend. Another type of learning environment consists of a runnable Scratch project that the students are asked to change until it shows a certain behavior. This type of task encourages students to test hypotheses about the effects of certain commands and to identify (and fix) misconceptions. Taking the concept of repetitions as an example, Jiří Vaníček illustrates how the curriculum was constructed. The goal of each unit is to generate a "generic model," a mental model that is decontextualized and so well understood that students are able to use it in new situations.

Learning and empowerment

« 8 » There seems to be a contrast between the constructionist idea of learning by creating personally relevant artefacts and a concept-oriented curriculum basically consisting of teacher-designed "etudes." However, even a mini-task that can be solved in minutes can be relevant. The context (the "story") is essential, because it can give an idea that the practiced concept might be useful to create something relevant. Not all examples presented in the article can offer a meaningful context. For example, the exercise "Let's shorten scripts" (Figure 8) is merely "technical." On the other hand, "Rocket track" – focusing on control structures – has the potential of creating a "feeling of empowerment": "If I understand this technique, I might be able to create a cool project." This is basically the idea of "skill cards" for game design. One side of a skill card shows an interesting feature of a game, and on the other side the programming is explained.

« 9 » Etudes are developed by professional educators, because the design requires specific expertise:

- Each task must be related to a relevant programming concept.
- There must be a reasonable order of etudes, starting with basic concepts and later covering more sophisticated concepts.
- The Scratch implementation requires rather advanced techniques ("Make a block").
- Crucial for motivation is an appropriate level of difficulty. A task must be stimulating but not too difficult.

« 10 » In what way can students be involved in the task design? One of the main challenges for designers is to keep the tasks interesting. Children might be involved not only by creating sounds and images, but by finding exciting contexts for tasks: stories that are highly relevant for them and that are reasons to learn programming. So let me conclude my commentary with a question for the target author: Are there ways to challenge pupils' creativity in the context of concept-oriented tasks design? (Q1)

References

Kafai Y. B., Peppler K. A., Chiu G., Maloney J., Rusk N. & Resnick M. (2009) From Photoshop to programming. In: Kafai Y. B., Peppler K. A. & Chapman R. N. (eds.) The computer clubhouse: Constructionism and creativity in youth communities. Teachers College Press, New York: 136–144

Lakoff G. & Núñez R. E. (1997) The metaphorical structure of mathematics: Sketching out cognitive foundations for a mind-based mathematics. In: English L. D. (ed.) Mathematical reasoning: Analogies, metaphors, and images. Lawrence Erlbaum Associates, Mahwah NJ: 21–92.

Papert S. (2000) What's the big idea? Toward a pedagogy of idea power. IBM Systems Journal 39: 720–729.

Resnick M., Maloney J., Hernández A. M., Rusk N., Eastmond E., Brennan K., Millner A., Rosenbaum E., Silver J., Silverman B. & Kafai Y. B. (2009) Scratch: Programming for everyone. Communications of the ACM 52(11): 60–67.

Michael Weigend studied computer science, chemistry and education at the University of Bochum and the University of Hagen and received a PhD in Computer Science from the University of Potsdam, Germany. His main research interest is media design for computer science education. He is a teacher at a secondary school in Witten, Germany and he has taught Didactics of CS at the University of Hagen for almost 20 years. He has published several books on computer programming, web development and visual modelling.

Received: 27 May 2019
Accepted: 7 June 2019

Author's Response

Programming as a Playground for Computational Thinking Development

Jiří Vaníček
University of South Bohemia, Czech Republic • vanicek/at/pf.jcu.cz

> Abstract • Programming is more a tool for pupil development than an educational goal. A description of the context of the creation of the textbook for programming could help to answer some questions leading us too far astray from the theory of concept-building programming education described in my article.

« 1 » In the commentaries on my target article (for which I thank the commentators), I found two basic sets of questions. The first one touches upon the methodology of programming education using concept building itself. The second one deals with the educational materials, the curricula created using the approach described in the target article and its implementation. These questions would be easier to answer if the context were known in which these materials were created. The target article had a methodological orientation and hence did not describe in detail some aspects of the project "PRIM" (Support for Development of Computational Thinking) within which framework these educational materials were developed. So, I think that describing the context of the recent changes in informatics education in the Czech Republic could help to better understand my answers.

A textbook of programming as a small piece of a big change

« 2 » The PRIM project has been crucial in the transformation of information and communications technology (ICT) education in the Czech Republic. The main goal was to change from a purely user-oriented approach to include the basics of computer science, and the PRIM project has to prepare the conditions for these changes in the field of computer science. Among other things, the PRIM project has to create educational materials that cover the new computer science curriculum for all school levels; from kindergarten to upper high school.

« 3 » Eleven new textbooks with an average size of 20 lessons have been created in the frame of the PRIM project. One of them is the textbook for programming in Scratch (for pupils of about 12 years of age) described in the target article. The authors endeavored to create this textbook in way that followed on from the textbook for primary programming (which, in turn, was inspired by the textbook created in the Scratchmaths project, UCL 2016). Their goal was to make it suitable even for absolute beginners without any previous programming experience. This textbook is followed by the textbook for more advanced pupils, containing suggestions and teaching support for larger programming projects.

« 4 » Along with the textbooks on algorithmization and programming, "unplugged computer science" textbooks were also created in which pupils are introduced to topics such as modeling, coding, information systems, and working with the data by taking part in unplugged activities and solving Bebras challenge contest tasks. Another area covered by these textbooks is the basics of robotics, in which the youngest play with the robotic Bee-bot toys, and the older pupils with robotic building sets (Lego, WeDo, Mindstorms) or with embedded systems such as Micro:bit or Arduino.

« 5 » Now let me turn to **Ágnes Erdősné Németh**'s Q1. Scratch has been chosen as the programming environment for its quality, availability and versatile connectivity. Scratch, with its "wide walls" (Resnick 2009), allows plenty of varied activities of sprites that can be programmed, e.g., drawing simple pictures and more complex ornaments, writing words by stamping letters, animations and movement of sprites, interviews of sprites and storytelling, simulations, composing music, preparing quiz tasks and the creation of simple computer games. Scratch is available online for free – which makes it additionally attractive for chronically underfinanced Czech schools. Scratch has also many extensions to control robotic and embedded systems like Lego WeDo, Mindstorms, and Micro:bit. So, the same environment can be used for working with different robotic systems without it being necessary to learn a new environment.

« 6 » The Czech national curriculum does not dictate which programming languages to teach. Rather, it is up to the teacher to choose the appropriate environment, language and curriculum for pupils to attain the required competencies and develop computational thinking (**Erdősné** Q2). The teacher can also use some materials outside of the project, a self-made curriculum, or decide when not to push forward with the curriculum and when to stop. At the time of writing, we still lack compulsory computer science education for primary and lower-secondary schools. Its introduction is expected in the next three years, so I can only write about what has been prepared in the context of the PRIM project. After this course, pupils can create complex programming projects in Scratch using the more advanced programming textbook, or move onto Python language with the use of the textbook "Programming in Python" for upper-high schools, and then onto programming Micro:bit using Python according to another prepared textbook.

« 7 » Erdősné is interested in tasks for children outside the mainstream, for those who are gifted and those who are falling behind (§13). Some of the activities used, such as discussion or inquiry, require pupils to work together, but some of the activities require a more individual tempo, so some individualization is necessary. This individualization is achieved by providing a textbook with both a basic and expanding curriculum organized in the shape of a "tree," where the basic curriculum is placed within the tree trunk. More advanced pupils turn their attention from the trunk to the branches, where they can work on more complex explorative or problem-solving tasks without it being necessary to use new concepts. Meanwhile, the other pupils move along the trunk, and the teacher can easily return those pupils working on tasks in the branches to the mainstream so that all the pupils can then move onto the next educational concept together.

« 8 » As I wrote above, textbooks for the basics of CS that include topics such as modeling, coding, or information systems, created in the PRIM project, systematically use Bebras challenge tasks (http://www.ibobr.cz). However, we cannot use "beaver tasks" in the textbook for programming in Scratch in their current form, because such tasks require platform independence and thus are not put into a specific programming environment. Nonetheless, themes and ideas from "beaver tasks" were used in the discussion tasks.

« 9 » Maciej Sysło disagrees with the suggestion that hard "beaver tasks" are difficult to solve without an understanding of the concepts related to programming. He argues that some of the contest participants are successful without programming knowledge (§5). However, "beaver tasks" are concept-based tasks (Dagienė, Sentence & Stupurienė 2017: 24) and one of the criteria for a good "beaver task" is that it is independent of the school curriculum (Dagienė 2011: 64), so tasks have to avoid possible pre-knowledge of contest participants. Contest organizers suppose that some participants will be successful without having been taught the necessary concepts for solving these tasks. In the Czech Republic we have the same experience as Poland, where there are participants with the highest possible score who had no programming education at school. However, we do not know whether they are genuinely unfamiliar with programming (because they could program outside of school) or whether these winners are simply geniuses, because the ratio of such excellent participants is very small (in recent years less than 0.1% in upper-high-school categories in our country).

« 10 » To respond to Syslo's reaction to the requirements of piloting teachers in §53, we are aware of the risks of teaching programming without previous practice. One of the main goals of the PRIM project is to prepare and pilot a system of intensive in-service education for primary and lower-secondary teachers so that they will be able to program at the basic level and teach the basics of programming in Scratch using textbooks created in the project. We have to prepare textbooks, though, that are suitable even for teachers who have to teach this topic without attending these courses or without any previous knowledge; thus they will be completely dependent on the information and instructions in the textbook. The second reason for choosing teachers without knowledge of programming for piloting was to easily spot the weak areas in the textbook and to focus our attention on these parts when preparing the module of educational courses for teachers. I would also add that the first piloting was carried out using very experienced teachers.

Programming as a tool

« 11 » As the title of my response emphasizes, we do not consider programming skills to be the final goal of education; we do not want to produce programmers. We see programming as an excellent playground in which children can develop their computational thinking (with an emphasis on thinking) and learn to understand how a computer works by solving algorithmic and programming tasks. So, the answer to Sysło's Q1 is that we do not prepare textbooks and curricula for the teaching subject of "programming," but for programming and algorithmization as a topic of the compulsory CS curriculum.

« 12 » Sysło's motto is "First think computationally then program" (§1), which he perceives to be the opposite of my approach described in the target article. The difference is that his motto describes the situation when solving a problem, whilst I am describing the process of learning. In a way, our motto could be "First play and then think about it." Our approach is first to learn the language and then to use it for solving problems. Even in algorithmic environments for small children like Scratch Jr (https://www.scratchjr.org), Emil (https://www.robotemil.com) or Thomas The Clown (http://www.skolkahrou.sk/vylety-sasa-tomasa), once one leaves direct manipulation mode there is simple language that must be grasped first to begin to solve tasks.

« 13 » Chantal Buteau is interested in which arguments to use in order to persuade teachers to change their educational style (§3). I can answer that in this way: The best form of persuasion to emerge from the pilot is personal experience: when a teacher tries to teach according to our textbook and recognizes that it works, that pupils can learn programming when he follows this approach. One advantage is that programming is a new topic for most of the teachers who have not taught it before, so they do not have any ingrained habits regarding how to teach this topic in a different and more "traditional" way. They could take this approach of "active play" as specific to the field of programming. The quality of the educational materials will persuade teachers to follow this approach. Thus, the first chapters of our textbook were created in a way that does not provide any support for the traditional approach of reading, issuing instructions, showing presentations, giving out answers and drilling pupils. Discussing and discovering a task is hard to omit because the tasks follow one another. Teachers are repeatedly instructed in the methodological materials not to assess pupils' answers during discussions. Instead they should moderate the discussion and allow the computer to decide what is correct in cases where there is no consensus in the classroom. The implementation of this approach during in-service teachers' preparation could serve as a good argument for accepting it. Also, we should not forget that when starting to learn programming a teacher's self-confidence is rather low, so if she sees that she can learn to program in this way there is a chance that she will use this approach when teaching.

« 14 » Buteau (Q1) brings up the practical question of whether teaching materials prepared in a constructivist way could enrich pupils' learning of programming even if it is taught in a traditional way. This question gives rise to another question: whether teachers who do not know how to teach in a constructivist way can apply this approach by following methodological instructions in the textbook alone. This is hard to answer without further research. In our particular situation, we do not have many teachers teaching programming to this age group of pupils so we cannot generalize. During piloting we observed that teachers sometimes tended to depart from the required approach, e.g., they offered up the final knowledge first instead of allowing the children to discover it. One teacher who fought with this tension for months became suddenly aware that it was more comfortable to change the climate in the classroom to what he described as "children's club."

« 15 » My answer to Buteau's Q2 on my position regarding the importance of exploiting other dimensions of pupils' computational thinking is as follows: The principles described in §26 are primarily connected to the first dimension of computational concepts. But focusing on concepts does not exclude the application of other principles in the creation of a textbook. Considering that (in the special case of our project) our textbook is dedicated to teachers without programming practice – giving them very little experience to develop computational perspectives of children – the activities have to contain such a dimension. The activities have to show children (and we tried to fulfill this) what else it is possible to program, and also that they can express themselves by creating their own or modified stories or graphics.

« 16 » When applying the computational thinking perspectives of Karen Brennan and Mitchell Resnick (2012: 10), it is necessary to keep in mind the different organization of "programming education" in the classroom and in the Scratch community. For example, the perspective of "connecting" happens more naturally in the classroom, where pupils can work in pairs and discuss things in direct communication rather than via the online communication tools Scratch offers. In accordance with Sysło, who wrote in §6 that learning programming concepts should be independent of the environment, we use the Scratch environment as little as possible. Thus, we do not use online comments and other community tools. For the same reason, we have prepared didactical environments to avoid wasting time in non-productive activities such as checking costumes of sprites, background or sound galleries. We need to spend the given amount of time on programming and we know that non-programming activities, like choosing, importing and graphically editing appropriate costumes for sprites or the background of the stage, are included in the textbook for creating advanced programming projects, which gives enough scope for this.

« 17 » I share Sysło's opinion about the Hour of Code activities (§7). We use them, but mostly for the purpose of motivation. We cannot use them for teaching algorithmization because they sometimes contain pedagogical mistakes such as big gaps between two consecutive tasks. For example, the activity "Coding with Anna and Elsa" (https://studio.code.org/s/frozen) uses *repeat block* for the first time in task #4, but the very next task, #5, requires the use of nested repeat blocks without providing any clue about how to use them. We observed that children who could not pass this task gave up on the whole activity. In this particular case, the role of the teacher as a helper is not an easy one. She can show the correct place to put the block, but the child has no opportunity to understand why.

« 18 » Michael Weigend draws attention to the risk of pupils becoming bored when they program in Scratch. We are aware of this risk, and motivation for such difficult work is very important. As I describe in part §26 of my article, each activity, however short, must have some effect. So, we tried to prepare a lot of varied activities: in one lesson children prepare a short comic story, in the next they go through a maze, then they fly by a rocket, next they simulate the movement of a fly in a bottle, and so on. Skilled pupils can continue to solve advanced tasks or to create more complex solutions (e.g., to draw more complex or colorful ornaments, try another way of controlling the rocket movement, to enrich the interactivity of a created game). The Scratch environment turned out to be very helpful because it offers many powerful commands for different types of activities.

« 19 » Not all activities in the textbook are creative or exploratory. If we have to teach children to read and understand the program code, sometimes it is useful to give them "technical" tasks such as "Let's shorten scripts" (Figure 8 in the article). These tasks have no big connection with programming itself, but they allow children to play the role of experts; they can guess, explain, and argue. Children enjoy communication – and unfortunately this is rare in Czech schools. We hope that these activities can help to introduce a new approach in terms of how to teach in our schools.

References

Benton L., Hoyles C., Noss R. & Kalas I. (2016) Building mathematical knowledge with programming: Insights from the ScratchMaths project. In: Proceedings of Constructionism 2016. Suksapattana Foundation: Thung Khru, Thailand: 25–32.

Brennan K. & Resnick M. (2012) New frameworks for studying and assessing the development of computational thinking. In: Proceedings of the 2012 annual meeting of the American Educational Research Association. Volume 1. AREA, Washington DC: 25. http://web.media.mit.edu/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf

Dagienė V. (2011) What kinds of tasks are good for contests? In: Proceedings of the 6th international conference on creativity in mathematics education and the education of gifted students, Riga, Latvia: 62–65.

Dagienė V., Sentance S. & Stupurienė G. (2017) Developing a two-dimensional categorization system for educational tasks in informatics. Informatica 28(1): 23–44.

Received: 28 June 2019
Accepted: 1 July 2019