Defining procedures in early computing education

Ivan Kalas[1,2], Laura Benton[2]

[1] Department of Informatics Education, Comenius University, Bratislava, Slovakia
[2] UCL Knowledge Lab, UCL Institute of Education, 23-29 Emerald Street, London, UK
kalas@fmph.uniba.sk, l.benton@ucl.ac.uk

Abstract. From the early years of educational programming researchers considered procedural abstraction a key instrument of computational thinking and tried to understand the cognitive difficulties encountered through this concept. Defining procedures is promoted in renewed computing curricula in several countries. And yet, it is rarely acknowledged by more recent educational research. In this paper, we consider the fact that the delayed implementation of a mechanism for building procedures (known as *definitions*) within Scratch, a widely used programming environment for children, may have negatively impacted the focus within curricular content on this powerful idea.

In our research, which is a part of a broader ScratchMaths (SM) research project, we set out to explore which factors play a role in upper primary pupils understanding and utilizing the concept of defining procedures as a common and inherent instrument of their programming. We present our observations from the project design schools and demonstrate how they guided the development of our *SM pedagogic strategy for definitions*.

Keywords. primary computing education, procedure, abstraction, ScratchMaths

1 Background

Computer scientists have recognised the power of defining a *procedure* since the early days of computer programming in the late 1950s, technically defining its role as offering "a single point of reference for some small goal or task that the developer or programmer can trigger by invoking the procedure itself" [1]. Later, whilst studying the idea of a computational process in the 1980s, Abelson and Sussman [2] in their seminal writing identified three basic mechanisms of a programming language, including *the means of abstraction* by which compound elements can be named and manipulated as units[1], i.e. "... the means that the [programming] language provides for combining simple ideas to form more complex ideas" [ibid].

In the early years of educational programming, within the context of Logo programming Papert [3] proposed the metaphor of "teaching the Turtle a new word" to represent the process of programming a computer. In 1974 Perlman, inspired by Papert, tried to implement the concept of procedure in a tangible programming interface for preschool children in her Tortis Slot Machine, see [4,5,6]. Through this

[1] the other two mechanisms being primitive expressions and the means of combination

work she started considering the *cognitive difficulties* behind some aspects of programming [5, p. 4] after observing children becoming overwhelmed when introduced to multiple new concepts through her system.

For Papert, turtle geometry provided an excellent opportunity to practice "the art of splitting difficulties" [3, p. 64], for example through drawing a house by splitting it into two parts – a square and triangle. He proposed that a Logo procedure can become something named, manipulated and recognised; terming it "an object to think with".

Since the early 1980s the mechanism of defining new procedures (in different forms) has been implemented in many programming environments for novice programmers, including children[2] and this is promoted through renewed computing curricula in several countries highlighting the power of abstraction and procedure. For example, Computing at School (CAS) in the UK [7] characterises procedure as a mechanism of abstraction, an instrument of generalisation, a pattern to be used to control complexity by sharing common features, and suggests "these abstractions may be deeply nested, layer upon layer*"* [ibid, p. 11]. CAS recommends that as well as *using procedures* pupils should become proficient in *creating new abstractions* of their own.

1.1 Defining Procedures in Research Literature

Within the Logo culture of 1980s and 1990s, researchers frequently examined the *procedural thinking* of the learners, but their studies often point to some inherent difficulties with the notion of procedure, see e.g. [8-10]. Pea et al. [10] observed that most of the pupils involved in their study in the early 1980s did not spontaneously accept programming practices such as "... structured planned approaches to procedure composition, use of conditional or recursive structures" [ibid p. 211].

Despite the legacy of being difficult and not naturally exploited by children as an everyday instrument in their programming, Logo educators and reserachers have always considered procedural abstraction to be one of the most 'powerful ideas' of computing education. However, this is rarely acknowledged by more recent educational research.

Most of the research projects looking at the learning of computer science and computing concepts focus on *variables*, *loops*, *conditions* and *control structures*, *message passing* or *concurrency*, often not mentioning definitions at all. For example Meerbaum-Salant et al. [11] study how Scratch can be used to teach computer science, focusing on 'standard' key concepts, but not on defining new blocks. Similarly, Ouahbi et al. [12] study how novices learn basic programming concepts by creating games but do not include definitions of new blocks in their observations.

Vaníček [13] identifies several potential risks in the emerging Scratch programming practices of the student-teachers, including unnecessarily long scripts. However, defining new blocks is not considered among the instruments to cope with that risk.

Futschek and Moschitz [14] explore a transition from a playful programming environment with tangible objects to a virtual Scratch environment and identify five basic

[2] such as Karel the Robot (1981), Solo (1983), Boxer (1986), Roamer (1989), Show and Tell (1990), Turingal (1991) among others, see e.g. [4]

computational concepts which should be present in learning scenarios with the aim to develop early algorithmic thinking, abstraction being one of them. They suggest that learners should experience a transition from perceiving a basic virtual command as an abstraction of a basic tangible action to perceiving a command as an abstraction of a compound (constructed) action.

1.2 Defining Procedures in Scratch

The Scratch programming environment has become an icon of the recent widespread interest of schools around the world in computing education for every pupil. It is a visual programming environment that allows users "to learn programming while working on personally meaningful projects such as animated stories and games" [15].

Brennan and Resnick [16] explain that in Scratch abstraction is employed at multiple levels "from the initial work of conceptualizing the problem to translating the concept into individual sprites and stacks of code". However, the previous version of Scratch (1.4) had no means to employ abstraction by defining new procedures. In 2010 Maloney et al. [15] wrote:

"Early versions of Scratch had a mechanism for creating procedures. In early field tests, however, many users were confused by procedures since they seemed very similar to broadcasts – both involved associating a name with a collection of commands. In the interest of simplicity and minimalism, procedures were removed from the language before Scratch was officially released..." [3]

More recently in Scratch 2.0 (released in 2013) the functionality for defining procedures was implemented as the **Make a Block** operation (see Fig. 1).

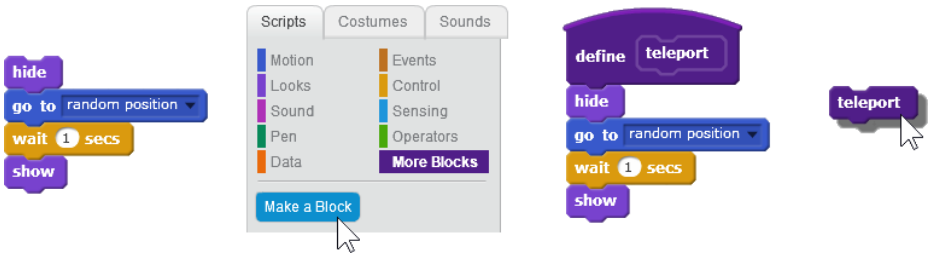

Fig. 1. In Scratch the **define** hat block is attached to a script, thus defining a new block

2 Procedural Abstraction within ScratchMaths

The research reported within this paper is a part of a broader project, ScratchMaths (SM), which aims to explore connections between developing computational and mathematical thinking in the upper primary age pupils[4] in England [17,18]. In the project we have iteratively designed detailed curriculum materials for computing

[3] later on we probe this observation through our own experience with the SM schools
[4] i.e. aged 9-11 years

lessons in years 5 and 6, which are currently being trialled in 50+ primary schools across the country. The intervention consists of six modules (three per year), with each module consisting of a series of activities organised within investigations (with accompanying classroom resources). The first three modules focused more on introducing key computing concepts (sequencing, repetition, algorithm, debugging, abstraction, initialisation, randomness, conditions, expressions, broadcasting), with links to mathematics made implicitly and the remaining three modules explicitly focused on particularly challenging mathematical concepts (place value, ratio and proportion, coordinates and geometry). Each year of the intervention included a two-day professional development program for class teachers, which was intended to introduce the 'big ideas' of the SM curriculum as well as the pedagogical approach to delivering the intervention. Through the research conducted as part of this project, we seek to better understand the *construct of procedural abstraction* in early computing education.

In their new framework for studying computational thinking, Brennan and Resnick [16] identify three key dimensions: *computational concepts* (like sequences, loops, events etc.), *computational practices* (like testing, debugging, reusing, remixing etc.) and *computational perspectives* (about the world around us). In our research we extend their first dimension into *computational constructs,* which comprise *computational concepts* (like procedural abstraction in this case) and *computational procedures* associated with the practice of the learners to exploit the concept.

2.1 SM Pedagogic Strategy for Definitions

Through the design of the SM intervention we recognise five implicit stages in developing the construct of procedural abstraction:

1) Perceiving a script[5] as an object to work and think with: One of the key SM design principles is systematically building the distinction between:

- *direct manipulation*, e.g. dragging a sprite (a programmable object) by the mouse or switching its costume (its appearence) by clicking on a different costume in the list.
- *direct drive*, clicking an isolated block (command) in the scripting area, thus getting an immediate and unambiguous basic reaction, e.g. clicking **move -50 steps** block would make a sprite move backwards 50 steps.
- *computational drive*, building and using a script as a representation of the compound future behaviour of a sprite e.g. by clicking it in the scripting area. Typically it is only later that a script (a behaviour) is turned into a complete reaction to a certain event through the addition of a hat block[6], e.g. **when this sprite clicked** or **when green flag clicked**.

[5] in Scratch a stack of blocks snapped together, a piece of program

[6] the topmost block to start a script, e.g. **when this sprite clicked** hat block

By adopting this distinction pupils start perceiving scripts (with no hat blocks, see the lower left script in Fig. 2) as *patterns of actions*[7], representations of partial or complete *behaviours*, as objects to build, explore, modify, and use (i.e. as objects to think with) and possibly abstract later. In our intervention we incorporate activities developing this approach as a preliminary phase for developing procedural abstraction.

2) Giving a name to a script. When a useful script has been built it can be given a name (Fig. 2 – right). Within our SM pedagogical framework [17] we encourage pupils to follow the procedure: (i) build a script, debug and use it, (ii) give it a name, i.e. attach a new **define** hat block to the script, (iii) keep the define script (the definition), and (iv) use the new block as a shortcut instead, as a name of that pattern of action – using the defined block in isolation (in the direct drive mode), then within a script (in the computational drive).

In SM Module 1 pupils define more new blocks for creating different visual patterns and combine them in short scripts to draw complex circular patterns. While doing that, in line with our pedagogical framework we suggest class discussion points to *explain* and *exchange* ideas such as: *How did you teach a sprite a new command? Why did you make new blocks? How did it help your programming, thinking, and problem solving? What name did you give your new block and why?*

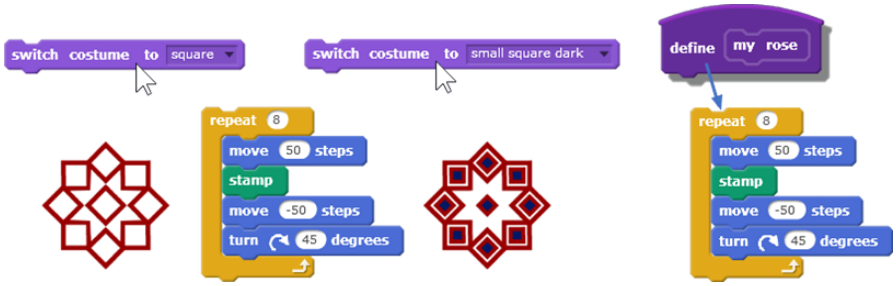

Fig. 2. Naming a useful script – i.e. certain *pattern of action*

3) Working with new blocks (own and provided): Pupils repeat the same process of making new blocks as useful shortcuts for previously built and used scripts in several contexts, to draw different compositions, e.g. a tower, a house, a swarm of colour dots etc. Gradually, they also start using their own new blocks to build more compound definitions (called nested definitions).

Pupils also use new blocks created by the SM designers and available within specific 'starter' projects using within the modules, e.g. **set random pen size** (see Fig. 3), within their own scripts.

[7] in the context of SM Module 1, *patterns of actions* are procedural representations of the corresponding *visual patterns*

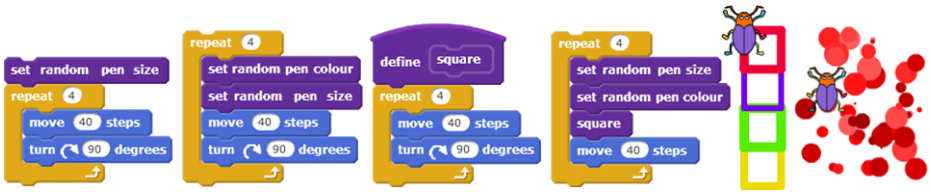

Fig. 3. Defining, using and modifying new blocks in different contexts

4) Customising and duplicating definitions. In the SM intervention there are contexts (situations) which require pupils to modify the behaviour of the pre-defined new blocks. They discover the "hidden" definitions[8], explore the input values to their **pick random ... to ...** operator blocks and modify them so that the defined blocks suit their design plan.

In another context, when working with multiple sprites, pupils also discover that each definition belongs to only one particular sprite[9] and that they need to copy or reconstruct it for use with other sprites.

5) Generalising definitions by indirect parameter. Pupils use the **ask/answer** pair of blocks, first using **answer** as an input to simple scripts, later also in the definitions of their own blocks as their *indirect parameter*, (Fig. 4 – left). Once they need to refer to several previous answers, variables are introduced and subsequently used in scripts and definitions (Fig. 4 – right).

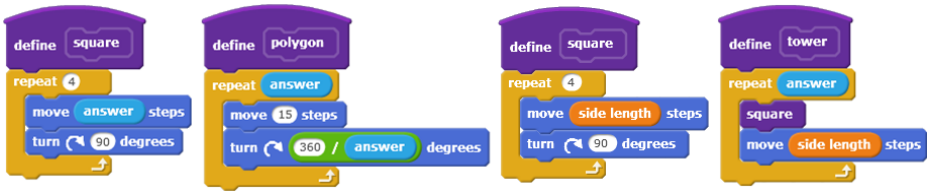

Fig. 4. Definitions generalised by using **answer** and variables as *indirect parameters*

3 Method

In developing the SM intervention, we followed a design research process to iteratively design the curriculum content and learning progression [17]. This involved drafting learning activities and subsequently trialling them with classes of pupils in one of four 'design' schools (primary schools in London). One to three SM researchers observed the lessons, took detailed notes as well as collected pupils' Scratch projects. The researchers then discussed together the observations and outcomes of each lesson, which informed the following redesign of the learning activity.

Within this paper, we focus on two specific research questions, which we explored through our design process:

[8] their definitions are so far "hidden" from view on the far right of the scripting area
[9] or to the stage, for completeness

RQ1: *Which factors play a role in (upper primary) pupils' understanding and choosing to utilize within their own programs the construct of procedural abstraction?*

RQ2: *Which computational procedures need to be mastered to support pupils' understanding and exploiting procedural abstraction (i.e. defining and using new blocks in Scratch)?*

3.1 Analysis

During the prototyping phase of the design research process each iteration of the curriculum content was trialled in three to five classes of the design schools, (between January 2015 and July 2016) with one or two researchers observing the lessons and collecting the Scratch projects. All activities were trialled in at least one school, with any activities that required modifications then retrialled in the same class (where substantial changes had been made) or a different class/school (where more minor refinements had been made). Further refinements were also made as a result of feedback received during the professional development sessions conducted with class teachers prior to trialling the final intervention within a wider group of schools. During this phase we focused on the systematic and coherent integration of procedural abstraction in all six modules of the intervention.

In this paper, we focus on analysing collected observations and projects from the perspective of the construct of procedural abstraction. Firstly we conducted a content analysis on the Scratch projects collected in several design schools during one Module 2 activity (partway through the year 5 curriculum), requiring pupils to define and use a block to draw a square, to identify the common initial issues or misconceptions the pupils encounter when learning about definitions (Research Lesson 1).

Secondly we conducted a content analysis on the Scratch projects collected during a 90 minute lesson in one design school which took place at the very end of the design research phase (after the trial of the year 6 content) to assess the pupils' understanding of the key constructs of the SM intervention (Research Lesson 2). This analysis was intended to identify the choices made by pupils with regard to the use of definitions within an open task following the SM intervention.

3.2 Initial Issues and Misconceptions

We identified three key issues in pupils' initial *building* and *use* of definitions during Research Lesson 1. Below we describe these issues and describe how they are addressed within our SM pedagogic strategy for definitions, which were subsequently promoted by researchers in classrooms later in the design research process. For example when defining a new block for drawing a square of the side length of 40, pupils:

i. did not attach the **define square** hat block to the script – new block **square** itself would then have "no behaviour" as it had no define script, i.e no definition.
   We advocate that teachers should encourage pupils to firstly build a script (stage

1) and then give it a name by creating a new block and attaching the hat block (stage 2).

ii. attached the **define square** hat block to the define script, however later started "stealing" the blocks from their define script, as if once having been defined, Scratch would simply "remember the definition"
We suggest that teachers should encourage pupils to get into the routine of moving the define script to the right of the scripts area out of the way once they are happy with it and then not touching it (unless they intentionally decide to modify it).

iii. attached the **define square** hat block to the define script, however continued building another **repeat 4 move 40 turn right 90 degrees** script whenever they needed to draw a square, instead of using the new **square** block as a name of that *pattern of action*.
We propose that teachers should encourage pupils to use their new blocks in isolation (stage 2) and then to use within different scripts (stage 3).

3.3 Use of Definitions after SM Intervention

During Research Lesson 2 the teacher first demonstrated the final behaviour in the full screen mode – so that pupils could not see the scripts of the model solution. When the (Beetle) sprite was clicked it asked how many houses it should draw and then drew a row of randomly sized and coloured houses. This could be repeated several times thus creating a hamlet, see Fig. 5.

Fig. 5. Picturesque hamlet, the final assignment in a design school

The class then as a group (in front of the interactive whiteboard) discussed the activity, steps and possible strategies, dealing with questions such as: *How many houses did the Beetle draw in one row and why? Do they all have the same side length? How are they positioned? How would the Beetle draw a house and a row of houses? Does the Beetle choose a side length, how does it remember that value while drawing a house? How will the Beetle learn how many houses to draw?* Pupils were not prompted to define their own new blocks.

A starter Scratch project was provided, with the Beetle sprite, a simple setup script (to clear the stage etc.), the **side length** variable already created, a pre-defined block **set random pen size colour shade** and two isolated blocks in the scripting area: **set side length to 0** and the **side length** reporter block.

Initial discussion took 15 minutes. Pupils were then divided into mixed ability pairs or threes by the teacher. Teams worked on their projects independently for 70 minutes with a short break, the teacher providing only limited guidance. There were

23 pupils in the class, and we collected 9 projects (which represented the work of 21 pupils), hereafter referred to as P1 to P9.

Our content analysis of the *nesting structure* of definitions within the projects was then conducted focusing on:

a) the definitions of new blocks and whether a new block is being used (nested) inside another definition – i.e. whether pupils achieved stage 3 of the SM pedagogic strategy for definitions.
b) whether the indirect parameters are properly implemented in the definitions – the **answer** block and the **side length** variable – that is, whether pupils achieved stage 5 of our strategy.

Fig. 6 presents the projects' nesting structure in the following way: the topmost triangular block represents the overall behaviour (solution), usually the **when this sprite clicked** script. Each circle represents a definition, the positioning corresponds to nesting, i.e using a new block inside another definition. For example, in P1 a block for drawing a **house** was defined, then used in the definition of a block to draw a **row of houses**, which is then used in the overall behaviour of the sprite. Sometimes the **house** block itself was defined by using another new block in it, usually a **square** (in P5) or a **square** and a **triangle** (in P6).

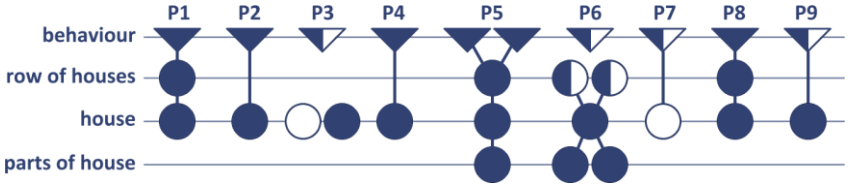

Fig. 6. Nesting analysis of the definitions in the projects

A circle or a triangle in Fig. 6 is filled, semi-filled or empty, depending on whether the definition correctly, partially or incorrectly[10] works with the indirect parameter i.e. **answer** or **side length**, the most advanced (stage 5) definition type in the SM curriculum.

4 Discussion

Despite some initial issues/misconceptions of definitions, our findings show that paying close attention to repeatedly exploring and explaining the practice of *building a script, giving it a name, keeping the definition and using new block as a shortcut* helped to reduce observed misconceptions and encouraged pupils to choose for themselves to utilise the power of definitions within their own scripts. Our pedagogic approach allowed pupils to automatize this computational procedure in different contexts, before creating a situation when pupils needed to get back to the define script and modify it.

[10] including not using it at all

Although back in 2010 Maloney et al. [15] reported certain confusion of definitions and broadcasts, we have not observed this within our research. It may be due to two factors: In the SM pedagogic framework [17] we strive to encourage pupils to work with (incomplete) scripts as (partial) representations of action or behaviours. A hat block is usually added only later, as an instrument to clarify how this behaviour will be activated. From the first module pupils add the **define** hat block to some scripts to give them name. Much later, in the third module (in year 5) they start using some scripts as reactions to receiving a message.

Through our nesting analysis of the pupils' final assignment, see Fig. 6, we looked at whether making new blocks has been adopted by the learners as an instrument to cope with complexity. We noted that every team made at least one new block and 5 of 9 teams made two or more new blocks (up to five). Two teams nested their definition in two levels, 2 teams even in three levels. We also noted that 8 teams correctly or partially correctly worked with the indirect parameter(s) in their scripts thus achieving stage 5 of the SM pedagogic strategy for definitions.

Through our content analysis we also identified that definitions of new procedures play **different roles** in the SM intervention:

- *Aggregating basic commands into one*: when several basic commands are simply attached together, with new command often carrying the names of its 'atoms', e.g. **set random pen size colour shade** or **dot stamp jump**.
- *Extending the language:* (in the sense of Abelson and Sussman [2]) when a new command gives a name to a compound reaction or behaviour, thus building a higher layer of the means of expression – abstracting from the detail, e.g. **house** or **teleport**.
- *Transforming the language:* when a basic command is 'replaced' by a new one to be used instead, e.g. replacing **move 20 steps** by a new block **move one tile**.
- *'Patching' the language*: when new block 'completes' the same layer of the means of expression as offered by other basic blocks – extending the language in a 'horizontal way'. This may lead to more consistent code, see Fig. 7 which illustrates the definition of such block – **previous costume**.

Fig. 7. *Patching the language*. While **next costume** is a standard block, 'symmetrical' **previous costume** block can be defined to highlight the analogy.

5 Conclusion

Although several designers updated their Scratch 1.4 materials to illustrate definitions, rarely is this construct integrated and systematically exploited as a truly powerful idea. In the SM intervention, the thread of developing procedural abstraction winds through all six modules, through five implicit stages. In our reserach we

acknowledge the importance of the role that definitions play in developing early computational thinking, facilitating [19] **decomposition** (by creating a structure, breaking down a problem), **abstraction** (by hiding detail), and **generalisation** (by highlighting certain patterns of action and encouraging to use them later in different contexts). Our experiences in the context of the SM intervention validate the importance of exploiting a tool with affordances that support pupils in building definitions, but also the importance of employing a pedagogic strategy that systematically develops all computational processes associated with the practice of the learners to exploit this concept.

Acknowledgments. The authors would like to thank the other SM project team members Richard Noss, Celia Hoyles, Piers Saunders, Johanna Carvajal and Dave Pratt, the Education Endowment Foundation for funding this work, and also all the teachers and pupils from our design schools for their invaluable contributions to the design and development of the SM intervention.

References

1. Technopedia: What does Procedure mean. Accessible online www.technopedia.com
2. Abelson, H., Sussman, G.J. with Sussman, J. (1985). *Structure and Interpretation of Computer Progams*. The MIT Press, 657 p.
3. Papert, S. (1980). *Mindstorms. Children, Computers, and Powerful Ideas*. Basic Books, New York, 230 p.
4. Kelleher, C., Pausch, R. (2005). *Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers*. ACM Computing Survey (CSUR), Vol 37 Issue 2, 2005, pp. 83-137.
5. Morgado, L., Cruz, M., Kahn, K. (2006). *Radia Perlman – A pioneer of young children cpmputer programming.* Current Developments in Technology-Assisted Education. Formatex, pp.1903-1908.
6. Perlman, R. (1976). *Using Computer Technology To Provide A Creative Learning Environment For Preschool Children*. AI Memo 360, MIT, 32 p.
7. Computing at School Working Group (2012) Computer Science: A curriculum for schools. *www.computingatschool.org.uk/data/uploads/ComputingCurric.pdf*.
8. Hillel, J. (1992). *The Notion of Variable in the Context of Turtle Graphics*. In: Hoyles, C., and Noss, R. (Eds.): *Learning Mathematics and Logo*. The MIT Press, pp. 11-36.
9. Leron, U. (1983). *Some problems in children's logo learning*. In Proc. of the *7th International Conference for the psychology of Mathematics Education*, Israel, pp. 346-351.
10. Pea, R.D. et al. (1985). *Logo and the Development of Thinking Skills*. In M. Chen and W. Paisley (Eds.) *Children and Microcomputers: Research on the Newest Medium*. Sage, pp. 193-212.
11. Meerbaum-Salant, O., Armoni, M., Ben-Ari, M. M. (2013). *Learning computer science cocepts with Scratch*. Computer Science Education, Vol 23 (3), pp. 239-264.
12. Ouahbi, I. et al. (2015). *Learning Basic Programming Concepts By Creating Games With Scratch Programming Environment*. Procedia – Social and Behavioral Sciences 2015.
13. Vaníček, J. (2015). *Programming in Scratch using inquirey-based approach*. In A. Brodnik, J. Vahrenhold (Eds.) Informatics in Schools. Curricula, Competences, and Competitions. Springer LNCS 9378. pp. 82-93.

14. Futschek, G., Moschitz, (2011). *Learning algorithmic thinking with tangible objects eases transition to computer programming*. In Kalas, I., Mittermeir, R.T. (Eds.) Informatics in Schools. Contribution to 21st Century Education. Springer LNCS 7013. pp. 155-164.
15. Maloney. J., Resnick, M., Rusk, N. Silverman, B., and Eastmond, E. (2010). *The Scratch Programming Language and Environment*. ACM Transactions on Computing Education, Vol. 10, No. 4, Article 16, 15 p.
16. Brennan, K., Resnick, M. (2012). *New frameworks for studying and assessing the development of cimputational thinking*. In *Proc. of the 2012 Annual Meeting of the American Educational Research Association*, Vancouver, Canada.
17. Benton, L., Hoyles, C., Kalas, I., and Noss, R. (2017). *Bridging Primary Programming and Mathematics: Some Findings of Design Research in England*. Digital Experience in Mathematics Education,, Vol. 3, Springer, doi: 10.1007/s40751-017-0028-x, pp. 115-138.
18. Benton, L., Hoyles, C., Kalas, I., and Noss, R. (2016). *Building mathematical knowledge with programming: insights from the ScratchMaths project*. In: *Constructionism 2016*, Bangkok, pp. 25-32.
19. Guttag, J. V. (2013). *Introduction to Computation and Programming Using Python*. The MIT Press, 298 p.