Racket Fun-ctional Programming to Elementary Mathematic Teachers

(Extended Abstract)

Dalit Levy[1]

[1] Kibbutzim College of Education, Tel-Aviv, Israel
dalit_lev@smkb.ac.il

Abstract. In light of current trends that regard coding as the literacy of the 21st century, the M.Ed program in mathematics education in our college of education offers its students – elementary mathematics teachers - two consecutive courses in computer science, in which they are exposed basic principles and practices of computing as well as its ties to mathematics and to mathematics instruction. In the last two years, both courses used the Racket functional programming environment. It is the aim of this presentation to reflect on the experience we had so far with having elementary mathematics teachers learn to design simple programs in a functional programming style.

1 Introduction

The M.Ed program in mathematics education at the Kibbutzim College of Education strives to ensure that students acquire extensive knowledge in the constantly changing field of mathematics instruction, based on research about how students think, the difficulties they encounter in learning math, new educational and psychological approaches and the integration of innovative technologies into education systems. In today's changing world with its constant advances in computer and communication technology, learning mathematics is more important than ever.

The program is tailored for experienced mathematic teachers at the elementary school, and seeks to enable them to integrate into the world of work and meet the demands of the 21st century. Coupled with professionalization, technological developments and the access to calculators and computers are enabling today's teachers to focus on developing mathematical thinking in addition to calculation skills as early as elementary school. Accordingly, it is important to teach and encourage teachers to use computers in mathematics classes in diverse ways, at all stages of learning. The program is intended to provide a response to this need on the graduate level. It is designed for teachers with bachelor's degrees in mathematics education or in mathematics with teaching certificates who wish to expand their knowledge and become leaders in the field of mathematics education in elementary schools.

One important aim of the program is to increase the students' knowledge of computers, computing and computer science and their relationship to mathematics. As many of the students begin their graduate studies in the program after many years of working as elementary school teachers and many years from completing their undergraduate studies, they often lack even the basics of the field of computing. When these students were asked in the beginning of the year "what is computer science", most of the responses were either "I don't know" or "Word, Powerpoint, Internet browsing". Few students mentioned programming but claimed to have no background in programming whatsoever.

In light of current trends that call for programming for all[1] and regard coding as the literacy of the 21st century, the program offers its second-year students two consecutive courses in computer science (CS). The first - "Basics of Computer Science" - is mandatory for all students of the program. The second – "Issues in Computer Science" – is an elective course. Both courses are designed for non-CS majors with no prior knowledge in programming, in order to give them an understanding of the principles and practices of computing as well as its ties to mathematics and to mathematics instruction. In the last two years, both courses used the Racket functional programming environment [1] and the 'Program by Design' approach [2]. Interestingly, after succeeding in the first mandatory course, most students continued to the second although not having to. It is the aim of this presentation to reflect on the experience we had so far with having elementary mathematics teachers learn, enjoy, and having fun with Racket functional programming tasks.

2 Course structure

Both courses integrate theory and practice while engaging students in the full cycle of design, based on the educational approach called 'Program by Design' [2,3]. That approach seems especially appropriate for a first programming course for non-CS majors, by focusing on developing design practices and desired programming habits from day one, by immersing learners into the profession's jargon, and by experiencing real coding with pictures and not only the conventional arithmetic jumpstart [4]. Although the students were experienced mathematics teachers, our preference was to start coding with pictures-as-objects and not with numbers-as-objects for two reasons. First, it presents a new "object to think with" [5] for those few who did have some programming background. Second, coding using prefix notation, as is the case in Racket functional programming, might seem more natural for these teachers that are so used to algebraic infix notation in their daily classroom instruction.

As is reflected in its website[2], 'Program by Design' is a longitudinal educational effort whose mission is to turn computing and programming into an indispensable part of the liberal arts curriculum both in high schools and undergraduate colleges. The curricula addresses some of the most well-known issues in introductory programming courses, like visualizing programming and thinking processes [6], emphasizing testing [7], and tailoring IDEs for learners' needs and prior knowledge by offering a series of pedagogic language subsets, in which at every level the error

[1] See Audrey Watters blog http://www.insidehighered.com/blogs/should-all-majors-not-just-computer-science-majors-learn-code (Jan 10, 2012). See also http://codeyear.com/

[2] http://www.programbydesign.org/overview

messages never depend on knowledge that the student does not yet have. Rooted in the paradigm of functional programing, 'Program by Design' is a functions-first approach to teaching introductory programming and problem-solving emphasizing good software engineering practices such as early testing from the beginning [8]. This is combined with the Racket IDE [1], featuring different language levels, simple syntax, customized error messages, and support for 'algebra of images' that enables students to write code for graphic- and animation-rich computing problems [4]. Most importantly, this educational approach offers design recipes to lead beginner students through a sequence of steps to obtain an understanding of the problem's nature and the solution program's behavior, hence the approach title. Testing is an integral part of the design recipe enabling test-first development [9]. Thus, the multi-step design recipe is useful not only to write programs but also to diagnose them. The design recipe scales naturally to the design of more and more complex systems of functions. This in turn empowers students to design programs for deep and interesting problems after just a minimum of introduction to the language, the environment, and the design recipes [2, 10].

An overview of the main computer science concepts in the first mandatory course is provided in Table 1. The course meets once a week for a four-hour lab session. The sessions are freely organized as a blend of short lecturer presentations, individual lab work, pair programming, and reflective class discussions. During the first weeks of the course, the students focus on fundamentals of programming by dealing with tasks involving pictures of their own choice, like overlaying one picture upon another, cropping certain parts of a picture, and drawing "the big picture" by combining picture parts [4]. Such focus is made available by the special library of functions that supports "algebra of images" included in the "beginner student" dialect of Racket IDE. Programming with pictures also enables students to creatively develop a small-scale programming project at an early stage of the course, like the flags mini-project on week 4. As it turned out, these creations also brought in the "fun" aspect of functional programming.

Table 1. "Basics of Computer Science" Course Structure

Week	**Content**
1	Introduction
2	Built-in functions for manipulating pictures
3	Global variables as names for complex functional expressions
4	Mini-project: Flags
5	Contracts and error tracking
6	Defining new functions
7	Parameters as local variables, scope

Since the students in the M.Ed program are not native English speakers, the various kinds of learning materials available on the internet as open educational resources were not very useful for the purposes of these courses. Therefore a special effort has been made to prepare appropriate learning materials such as worksheets, presentations, exercises, and tests in Hebrew. These materials have been made available for the students on the College's Moodle site as the course unfolded. As this is the first known case of using Racket and 'Program by Design' for teaching programming in Israel, these materials has been also used by some of the students after the course, when they got back to their schools as mathematics teachers.

While the first course only hints at elements of the design recipes, the second course starts with the idea of following a design recipe and compares it to similar ideas in mathematics problem solving models. The second course also meets once a week for 4 hours lab sessions, and goes further in designing test cases and more advanced programming, with numbers, words, pictures and animations alike.

One recurring pedagogical pattern is weaved into many of the learning experiences and the Hebrew curriculum materials. In coordination with the test-first approach [9] that play an essential role in developing the culture of ‘Program by Design’, students are constantly asked to predict what they would expect to happen in a given situation before running the program, and write it down; to carefully observe and check the results of running the program; and finally to explain the results (which may or may not be what they predicted). This learning pattern follows the very well-known Predict-Observe-Explain structure [11] found successful in science education. By using such pattern within the context of learning to program and by being able to embed it within the actual code, even mathematics teachers who are not going to be professional programmers have the opportunity to experience one of the key components of the professional process of program design even at very early stages.

3 Summary

A first course in computer science is in some sense an almost impossible task [4], in which students with various backgrounds need to learn (a) the grammar and the components of a programming language and (b) how to analyze a problem and design a program to solve it using that language. It is considered quite easy to get caught up in the details of (a) at the expense of (b), but it is better not to put the language itself as the focus of the study. The much more lasting knowledge is constructed through dealing with how to design a program that is both correct and easy to write, read, modify, and repair. This might be true for any first computer science course, whether in middle school, high school, or college. It is certainly true for an introductory course for elementary mathematics teachers who are not intended to become professional programmers, as is the case described above. The focus on design patterns, which are step-by-step "recipes" for getting from a vague description of a problem to a working computer program, gives such students the "taste" of how professional programmers work in real-life contexts. The emphasis on test-first tools and pair programming helps the students experience a culture of programming and thus better understand the professional discourse among teams of programmers.

Two groups of 25 elementary mathematic teachers each learned the courses so far, and three principles have proven viable for them. First, using pedagogically-grounded language tools and IDEs can indeed support novice learners' focus on the design process rather on the specifics of the language. Second, initiating the programming venture with tasks within the world of pictures, graphics, and animations, while building on "algebra of images" rather than on arithmetic, opens up a whole new and unprejudiced context for both novices and those who have had some previous programming experience. Within such a context, creativity often shows itself both at the level of the design process and at the level of the product, and adds joy and fun to the learning process. And third, applying test-first design and strict documentation requirements as early as possible (the third or four week) enforces disciplined programming while illustrating in a concrete way the program's desired behavior. Within the context of pictorial functional programming, students can combine small pieces of code into quite a complex program at an early stage. From the very beginning of the course they practice writing test cases before writing each function definition, and gain experience in phrasing each piece's contract [10], expected result, and test cases.

Finally, the proposed courses were indeed tailored for elementary mathematics teachers, but they might present a valuable alternative to consider for other kinds of students. Together with its ‘Program by Design’-based approach, such courses has the potential to give students an understanding of the principles of computing and knowledge about the practices of computing professionals. That knowledge will probably support students' ability to make the choice so much needed "In the emergent, highly programmed landscape ahead… - Program or Be Programmed" [12].

References

1. Flatt, M. and PLT: Reference: Racket. Technical Report PLT-TR-2010-1, PLT Inc. (2010). http://racket-lang.org/tr1/
2. Bloch, S., Clements, J., Felleisen, M., Findler, R.B., Fisler, K., Flatt, M., Proulx, V., Krishnamurthi, S.: Program by Design Project: Overview. Retrieved March 2013 from http://www.programbydesign.org/overview
3. Bloch, S., Proulx. V.K.: TeachScheme, ReachJava: Introducing OOP without Drowning in Syntax. J. of Computing Sciences in Colleges 22(4) (2007)
4. Bloch, S.: Picturing Programs: An Introduction to Computer Programming. King's College London: College Publications (2010)
5. Abrahamson, D., Howison, M.: Embodied Artifacts: Coordinated Action as an Object-to-Think-With. In: AERA 2010, Denver, May 3 (2010)
6. Lapidot, T., Levy, D., Paz, T.: Teaching Functional Programming to High School Students. In: Robson, R. (ed.), Proceedings of M/SET, pp. 245--249, San Diego, CA (2000)
7. Felleisen, M., Findler, R.B., Flatt, M., Krishnamurthi, S.: The TeachScheme! Project: Computing and Programming for Every Student. Computer Science Education 14(1), pp. 55--77 (2004)
8. Tuttle, S.T.: Introducing Programming in a Functions-First Manner, Using the "Program by Design" Approach. J. of Computing Sciences in Colleges, 27(1), p. 101 (2011)

9. Crestany, M., Sperber, M.: Experience Report: Growing Programming Languages for Beginning Students. In: Proceeding of ICFP '10, Baltimore, MD (2010)
10. Felleisen, M., Findler, R.B., Flatt, M., Krishnamurthi, S.: How to Design Programs, MIT Press (2001)
11. Linn, M.C., Eylon, B.-S.: Science Education: Integrating Views of Learning and Instruction. In: Alexander, P.A., Winne, P.H. (eds.) Handbook of Educational Psychology (2nd ed.) pp. 511--544. Mahwah, NJ: Erlbaum (2006)
12. Rushkoff, D.: Program or Be Programmed: Ten Commands for a Digital Age. Introduction, Retrieved December 2012 from http://www.amazon.com/Program-Be-Programmed-Commands-Digital/dp/1935928155#reader_B004ELAPME .