

Teaching programming to young learners using Scala and Kojo

Björn Regnell*, Lalit Pant[†]

*Dept. Computer Science, Lund University, Sweden, bjorn.regnell@cs.lth.se

[†]Kogics, Dehradun, India, lalit@kogics.net

Abstract—This paper presents an approach to teaching programming and abstract thinking to young learners using Scala and Kojo. Kojo is an open source IDE for the Scala programming language. The approach is based on Scala APIs for turtle graphics and functional pictures, a process of interactive exploration and discovery, and structured learning material that guides learners. The approach encourages playful self-learning of basic programming principles such as sequential execution, repetition, primitives, composition, abstraction, parametrized abstraction, and nested abstractions. It also includes tools to help children read and understand programs. Results from the use of Kojo and Scala in the teaching of young learners in Sweden and India are presented, along with a discussion of experiences and future development.

Index Terms—computer science education, first language, Scala, Kojo, turtle graphics, functional picture graphics

I. INTRODUCTION

This paper describes how Scala and Kojo can be used to teach programming and abstract thinking to young learners. Kojo is an open source Integrated Development Environment (IDE) for the Scala programming language [3], targeting beginners and young learners. The approach includes a step-wise introduction of basic programming principles such as sequential execution, repetition, composition, and abstraction. Experiences from the application of Kojo and Scala in teaching of young learners in Sweden and India are presented, along with a discussion of some advantages and challenges with using Scala as a first language for young beginners. The paper concludes with a list of ideas for further improvement of the pedagogical material and tools.

II. BACKGROUND

The Kojo project [2] was started in 2009, by its main contributor Lalit Pant, with the goal of providing an interactive learning environment for children in the areas of programming, mathematics, science, art, and more. Kojo combines a programming environment for Scala with graphics capabilities. The programming environment supports syntax high-lighting, code completion, code templates, evaluation worksheets, interactive program manipulation, and program tracing. For graphics, Kojo features a turtle graphics Application Programming Interface (API) inspired by Papert's LOGO and constructivist learning ideas [4]. The turtle API is combined with an API for functional picture generation and transformation that together can be used to make drawings and games with concise code. Kojo also includes features for interactive exploration

of geometry and algebra. Kojo is free software under the GNU GPL licence. Pull requests are invited on all aspects of the project, including the core framework, the graphics API, translation, and documentation. Learning resources are also open source. Kojo includes code from several other open source projects such as Piccolo2D [5], and Geogebra [1].

There are many programming environments aimed at helping young learners, including environments based on a programming language especially designed for young learners, such as Scratch [8], and environments using a general, professional programming language, such as DrRacket (based on a Lisp dialect) [7]. Kojo is, to the best of our knowledge, the first environment for young learners that is based on Scala.

III. APPLICATION




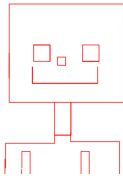



This section presents the application of Scala and Kojo in teaching in two cases – in Sweden (case 1) and in India (case 2). The contexts are described and some results are provided, together with examples of challenges given to young learners.

A. Case 1: Science Center LTH, Sweden

A project called "Programming for Everybody" (subsequently denoted PFE) started in 2012 at the Science Center LTH at Lund University. The general goal of the science center is to reach out to pre-university education and help to increase the interest in engineering education among the youth in southern Sweden. The science center opened in September 2009 and has since then had more than 121,000 visitors of all ages (as of January 2014). Visitors during weekdays range from school groups to company events and training. The center also takes bookings for a variety of different events and celebrations. It is open to the public on weekends and school holidays, when everyone is invited to try out interactive experiments, attend a show, or see an exhibition. Before 2012, our science center experiments focused on areas such as physics, chemistry, and electronics, but the area of computer science was lacking. The main goals of the PFE project are to (1) develop programming experiments for visitors with particular focus on school groups, and (2) to develop teacher training so that pre-university schools for all ages can help kids to discover the excitement and importance of computer programming.

Results. Within the PFE project, a Swedish turtle graphics API translation has been developed, together with a set of programming 'challenges' in Swedish – targeting kids of age

TABLE I
EXAMPLES OF CHALLENGES FOR BEGINNERS THAT PROGRESSIVELY INTRODUCE BASIC PROGRAMMING CONCEPTS.

Challenge	Image key	Solution	Programming concept
Your first program: forward		forward	Using an existing abstraction (procedure)
Draw a square using: forward; right		forward; right forward; right forward; right forward; right	Sequence
Draw a square using: repeat(4){ ??? }		repeat(4){ forward; right }	Repetition
Draw a human/alien/robot using: forward(100) right(180) left(180) hop(100) jumpTo(100,100)		//not shown for brevity	Using existing, parametrized abstractions (procedures with parameters) in sequence
Define your own, reusable square and use it to draw two squares: def square = ???		def square = repeat(4){ forward; right } square; hop; square	Creating a new abstraction (procedure)
Define a procedure that draws a stack of 10 squares: def stack = ???		def square = repeat(4){ forward; right } def stack = repeat(10){ square; hop } stack	Creating nested abstractions (a procedure calling a procedure)
Define a procedure that can draw squares of different sizes: def square(s: Int) = ???		def square(s: Int) = repeat(4){ forward(s); right } square(100)	Creating abstractions with parameters (a procedure with one integer parameter)

7 and upwards, with the only pre-requisite being the ability to read from a computer screen and use a keyboard and mouse. Since the PfE project started, we have had more than 7000 kids trying our programming experiments. More than 60 teachers have passed our programming courses comprising 2 half-days with assignments in between, to try out programming in class using Scala and Kojo. The introduction to programming to both young learners and to their teachers is based on a series of programming 'challenges' that covers a progression including sequential execution, repetition, abstraction, parametrized abstraction, and nested abstractions. These challenges have been iteratively developed based on feedback from kids and teachers. Table I illustrates how the progression of basic programming concepts are introduced through turtle graphics challenges, here translated into English. In Swedish, the solution to the challenge of drawing a square ("kvadrat") is: `def kvadrat(s: Heltal) = upprepa(4){fram(s); höger}`. This is enabled by Scala's ability to use `ää` in identifiers. We have iterated the Swedish turtle API translation, based on

feedback from kids and teachers, aiming at Swedish terms that are short and easy to write and spell. The complete Swedish translation and the programming challenges in Swedish are available on-line [6].

B. Case 2: The Kalpana Center and Himjyoti school, India

Kojo was introduced by Lalit Pant at Himjyoti school, Dehradun, India, in late 2009. Himjyoti is a school for bright, underprivileged girls from the state of Uttarakhand, India. At that time, Lalit was a volunteer teacher at Himjyoti. The motivation for the introduction of Kojo was a desire to introduce girls of ages 11 – 13 at Himjyoti to an interactive learning environment, where they could play with concepts in the areas of programming, math, and science. Kojo was used at Himjyoti till mid-2011 during one hour long activity classes that were carried out five times per week.

In mid-2011, Lalit started the Kalpana Research and Learning center at Dehradun, India, to provide exposure to computer based interactive learning to children in his neighbourhood,

and to serve as a research lab for his ongoing work with Kojo. One hour long activity classes are held at the Kalpana center three evenings a week. The teaching is based on a few key ideas – interactive exploration and discovery, trial and error, playfulness, and self-learning.


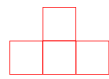
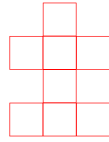
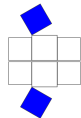
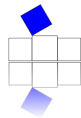

Results. More than one hundred kids have participated in the project since its inception – sixty for about six months, thirty for about twelve months, fifteen for about eighteen months, and five for more than twenty four months. All the kids were introduced to procedural and functional abstraction (via `def`) while working with the turtle API. Kids who worked with Kojo for more than a year were also introduced to the functional pictures API. Table II shows how the pictures API can be used to progressively build richer drawings using the basic programming ideas of primitives, composition, abstraction, and transformation.

IV. EXPERIENCES

Below, we have collected some of our experiences and reflections in applying Scala and Kojo in the previously described cases. We start with some of our general views on the application of Kojo and then continue with some of our views on Scala-specific matters.

- **The power of the turtle.** In line with Papert's original ideas [4], we have found the turtle metaphor to be a powerful tool in helping learners to understand programming through the act of drawing. Writing a sequence of commands to make the turtle create a desired drawing is a natural way to start learning how to program for most of the learners we have met.
- **Learning by trial and error with simple building blocks.** It is interesting to see how far learners can go in the process of making drawings and games using just a very simple subset of Scala (via `vals` and `defs`). Playing with working code examples that show something in action can help understanding significantly. This is facilitated by the quick edit-execution cycle in the IDE. The interactive program manipulation feature of Kojo also encourages learners to quickly try out many different ideas (using a slider), while program tracing helps them to gain insights into what the program does as it runs. If the teacher can help the learner to find this process of experimental coding joyful and rewarding, it is more likely that an eagerness to continue with programming is preserved.
- **Understanding abstraction.** For many learners, the introduction of user-defined commands (i.e., procedural abstraction) is an *aha!* moment, where they appreciate the power that has just been handed to them. Some learners, however, struggle to understand the significance of abstraction, and need to see more examples and to repeatedly experience the benefits of achieving more with less before they get it. We have found that the pace and sequencing of programming challenges and advancement in abstraction mechanisms need to be carefully adapted to each individual learner's current level. It is easy to lose confidence and interest when programs seem to become needlessly complex and do not run as you want. An important role of the teacher is to help young learners overcome such obstacles and at the same time find ways to demonstrate the utility of abstraction in concrete solutions to programming challenges that are interesting and relevant to the young learner. Real-world analogies are also helpful in getting learners to relate to abstraction.
- **Parameters and types.** As soon as kids try procedural abstraction with parameters, they run into the concept of types.

TABLE II
EXAMPLES OF CHALLENGES THAT PROGRESSIVELY INTRODUCE
PROGRAMMING CONCEPTS USING KOJO'S FUNCTIONAL PICTURES API.

Image key	Solution
	<pre>// Defining a primitive: def p = Picture { repeat(4){ forward(100); right(90) } } draw(p)</pre>
	<pre>// Using composition to combine // primitives; nested composition: draw(picRow(p, picCol(p, p), p))</pre>
	<pre>// Using abstraction to give names to // compositions; nested abstraction: def p2 = picRow(p, picCol(p, p), p) val p3 = picCol(p2, p2) draw(p3)</pre>
	<pre>// Using and composing transformations: def p2 = penColor(darkGray) -> picRow(p, picCol(scale(1.1) -> p, fillColor(blue) * rot(30) -> p), p) val p3 = picCol(flipX -> p2, trans(0, 5) -> p2) draw(p3)</pre>
	<pre>// Applying effects: val p3 = picCol(flipX * fade(250) * blur(2) -> p2, trans(0, 5) -> p2) draw(p3)</pre>
	<pre>// A more elaborate drawing: def star = fillColor(yellow) * penColor(noColor) -> Picture { repeat(5){ forward(150) right(720/5) } } clear() setBackground(Color(0, 149, 145)) val pic = picStack(weave(30, 10, 30, 10) * fillColor(red) * trans(-200, -200) -> PicShape.rect(400, 400), star, flipX * fade(140) * blur(2) -> star) draw(pic)</pre>

The slower learners have initial trouble with this, while the faster learners experience this as an opportunity to learn more. When they achieve the level of making and using their own parameterized abstractions, young learners often express the joy of having "done it myself". This in turn can be a strong driver for their internal motivation to continue learning, despite the efforts needed to overcome challenging concepts.

A. Advantages and Challenges of Scala as First Language

Next, we list some of our subjective reflections on the of using Scala as a first language for young learners. These reflections need to be investigated further, before any firm conclusions can be drawn on advantages and disadvantages of using Scala as a first language for young learners.

The *advantages* that we have perceived include:

- **A "real", general purpose, professional programming language.** We have found that many young learners, as well as many of their teachers, appreciate learning how to program using a "real" programming language – in the sense of a full-fledged, professionally used language, rather than a special language for kids.
- **Low floor – high ceiling.** Scala's concise, light-weight syntax and orthogonality of features makes it easy to start with a limited part of the language. Code can be "to the point" and the language does not get in the way as learners go about their work. As learners progress, more advanced abstraction mechanisms can be introduced step by step, in a carefully thought out progression. With Scala, challenges can range from pre-school to post graduate level, and learners can code at their level of fluency.
- **Wide walls.** The experimentation opportunities are wide due to direct access to the large base of existing, open source, Java libraries. This also supports the sense of being part of "real-world programming", which is motivating to many young learners as mentioned previously.
- **Flexible and pragmatic.** Scala is flexible [3] in the sense that it is up to the coder to make a pragmatic choice about what paradigms and idioms to use. The teacher can use the same language to contrast different paradigms and the learner can play with different idioms without having to learn another language. The optional operator notation also makes it easy to provide APIs, such as Kojo's turtle and pictures APIs, which act as concise sub-languages.
- **Native language coding.** Scala's support for UTF identifiers enables the Swedish letters Å, Ä and Ö in Kojo's turtle API. Learning to program in a foreign language can be difficult. Native coding enables not yet English-speaking learners to climb the thresholds of computational thinking without also having to deal with foreign concepts.
- **Catching bugs at compile-time.** Static typing gives feedback at compile time, helping young learners to catch bugs earlier, and reducing the risk of young learners giving up because of bugs in larger programs that are too difficult to track down at run time.

There are some *challenges* that we have faced when using Scala as a first language for young learners, including:

- **Difficult error messages.** Error messages are often difficult for young learners to interpret and make use of. These are expressed in compiler-oriented terminology, and the young programmer often has no hints on what needs to be changed in order for the program to compile. Learners are left to work around this by using IDE functionality to locate the

line in the code where the error is, and inspecting the source to determine the error.

- **Strange types.** A type annotation can be a double-edged sword. Types can get in the way of slower learners, while they facilitate increased learning for faster learners. Teachers may need to prepare simple explanations and use helpful analogies to convey what types are good for.

V. FUTURE WORK

We conclude this paper with a list of areas of particular interest to the further development of Kojo and Scala as tools for teaching programming, and more, to young learners.

- **Empirical studies on Scala teaching.** What works for which category of learners? What are feasible progressions? What subset of Scala is best at what stage? These questions, and more, would be interesting to study empirically with young learners as subjects.
- **More learning material for young learners.** Develop more learning material, e.g., on how to make rich art and fun games, with the help of math and physics, using Scala and Kojo.
- **Better error messages.** Provide simpler error messages with better hints on fixes. Investigate the opportunity of using language levels, as in Racket [7], which restrict available language features, while providing error messages that fit the learner's level of knowledge.
- **Improved gaming support.** Improve support for sprites, collision detection, and physics for more realistic games. Game-making engages young learners, and provides opportunity for learning mathematics, physics, and more advanced programming.
- **Hardware control.** Introduce support for physical computing using Arduino and the Raspberry Pi, as another area that has the potential to engage young learners in learning robotics and computer systems and understanding how hardware can be controlled by software.
- **More translation.** Kojo's menus are currently available in English, Swedish, French, Polish, and Dutch; the Turtle API is available in English, Swedish, Polish, and Dutch. Translation contributions are invited for more languages and more of the Kojo APIs.

Acknowledgments. The PfE project is partly funded by the Faculty of Engineering LTH, Lund University, Sweden. We thank all contributors to the Kojo project and all the wonderful kids who have helped us understand more about how to teach programming. Sponsors of the Kojo project include Kogics, Lund University, Typesafe and REACHA.

REFERENCES

- [1] Geogebra home page. <http://www.geogebra.org>.
- [2] Kojo home page. <http://www.kogics.net/kojo>.
- [3] Martin Odersky, Philippe Altherr, Vincent Cremet, Burak Emir, Sebastian Maneth, Stéphane Micheloud, Nikolay Mihaylov, Michel Schinz, Erik Stenman, and Matthias Zenger. An overview of the Scala programming language. Technical Report, IC/2004/64, EPFL Lausanne, Switzerland, 2004.
- [4] Seymour Papert. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc., 1980.
- [5] Piccolo 2D home page. <http://www.piccolo2d.org>.
- [6] Project 'Programming for everybody' home page. <http://www.lth.se/programmera>.
- [7] Racket home page. <http://racket-lang.org/>.
- [8] Scratch home page. <http://scratch.mit.edu/>.