2016

Uncommon Teaching Languages

Mark C. Lewis

Doug Blank
Bryn Mawr College, dblank@brynmawr.edu

Kim Bruce

Peter-Michael Osera

Citation

Mark C. Lewis, Douglas Blank, Kim Bruce, and Peter-Michael Osera. 2016. Uncommon Teaching Languages. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16). ACM, New York, NY, USA, 492-493.

Uncommon Teaching Languages*

Mark C. Lewis
(moderator)
Trinity University
1 Trinity Place
San Antonio, TX
mlewis@trinity.edu

Douglas Blank
Bryn Mawr College
Bryn Mawr, PA 19010
dblank@cs.brynmawr.edu

Kim Bruce
Pomona College
Claremont, CA 9171
kbb04747@gmail.com

Peter-Michael Osera
Grinnell College
Grinnell, IA 50112
osera@cs.grinnell.edu

CCS Concepts

•**Applied computing** → **Education;** •**Software and its engineering** → **General programming languages;** *Object oriented languages; Functional languages;*

Keywords

CS1; CS2; Programming-Languages

1. SUMMARY

Most courses of study in computer science begin with students learning to think algorithmically, and to express the solutions to problems using a programming language. The choice of *which* programming language is usually considered secondary to the choice of concepts, but the reality is that the vehicle we choose for teaching concepts shapes the way that students understand those concepts, and enables or inhibits the learning of certain concepts.

Most departments use one of a small number of mainstream languages that are well-established in industry and are backed by teaching and learning resources. A minority of departments choose to work with non-mainstream languages, finding that the advantages of those languages outweigh the disadvantages. What issues should we consider when choosing a programming language for our introductory courses?

A. Use in the "real world". A department may be pressured to use a language that is prevalent in local industry, local feeder schools, or standardized tests. For example, local companies might like students to be prepared for internships by being taught in whatever language they use for production. Unfortunately, "real world" languages often come with complex features, that, while useful on large industrial projects, make them unsuitable for students, particularly beginning students, and have a tendency to become less pure, more complex, and harder to teach over time.

B. Available resources. More teaching resources can mean less work for the instructor and can make life easier for students looking to answer questions. Unfortunately, it also means that cheating is easier and that students get lazy. After all, why figure out your own approach when you can find the answer on *StackOverflow*?

C. Pedagogical benefits. Does the language have a clear syntax that teaches the students the vocabulary of the programming paradigm? Does it have a straightforward semantics for expressing algorithms? Does it add to the essential difficulties of learning how to express algorithms its own set of "accidental difficulties" that distract students from learning the essentials?

The idioms natural to a particular language can encourage beginning students to develop specific programming habits. Does the language and its libraries encourage habits, such as clear interfaces, single-responsibility methods or functions, model-view separation, and test-driven development, that will serve students well regardless of language? Or does the language encourage techniques that don't transfer well to other languages?

D. Integration with the curriculum. Students need a path from the introductory language to languages used in later courses; this argues for using the same or similar languages. However, students should also be exposed to a variety of styles of language during their degree program; this argues for using radically different languages at different stages in the program.

2. OCAML (PETER-MICHAEL OSERA)

OCaml is a strongly-typed functional programming language derived from ML. At the University of Pennsylvania, we start our CS2 course in OCaml and then transition into Java in the second half. This approach seems to violate several of the desiderata presented in Section 1: OCaml is neither a common language found in industry nor elsewhere in our curriculum and there are comparatively few resources for teaching OCaml at this level. However, OCaml strongly supports the type-directed, test-driven design process and usage of immutable data structures that we emphasize in the course.

While OCaml possesses object-oriented features, we elect to use a small functional subset of the language which al-

SIGCSE '16 March 2–5, 2016, Memphis, TN, USA

ACM ISBN 123-4567-24-567/08/06…$15.00

DOI: 10.475/123_4

lows the students to immediately jump into the course and solve problems without being bogged down in syntax. Furthermore, the fact that OCaml is not widely used becomes a boon because it "levels the playing field" between our students who have a wide range of programming experience, particularly with Java. Finally, utilizing a functional programming language early in the curriculum allows us to draw meaningful comparisons between the object-oriented paradigm and other, lesser known paradigms.

3. GRACE (KIM BRUCE)

Grace [2] is by design an object-oriented language for novices, in the same way that Pascal was by design a procedural language for novices. It is simpler syntactically and semantically than Java, yet includes more powerful and expressive modern features. It provides better support for object-oriented features than Python. It omits many of the features that make industrial languages so hard for novices.

Grace supports two different graphics libraries and a library for supporting animations. There is also a well-designed data structures library. A draft textbook, *Programming with Grace*, based on the `objectdraw` library and event-driven programming, is freely available. Grace programs can be written and run in a web browser, so Grace can be run on all platforms just by visiting a URL.

Dialects (like those of Racket) can be used to add new constructs to the language or to restrict the language to allow students and instructors to focus on key concepts. Grace has a simple syntax that avoids most of the "noise" that pollutes code in other languages; for example, there is no `public static void main`, and the visibility defaults are exactly what novices require. Programs can be anywhere in the spectrum from completely dynamically typed to fully statically typed.

Grace was designed to make it easy for students to move on to languages like Java, Scala, and Python. In our experience, the only difficulty in shifting from Grace to Java is getting used to the weird special cases in Java that get in the way of the programmer. Picking up Python should be even easier. Many of Grace's features are similar to Scala, so we see a smooth path from Grace to Scala.

4. JIGSAW, SCHEME, AND PROCESSING (DOUGLAS BLANK)

Jigsaw is a drag-and-drop, block-oriented language designed to help transfer knowledge between languages from Scratch to Python. Jigsaw is one of the languages used in the Calico system [3], all of which can share libraries. For example, the same graphics or robot library can be used in Jigsaw and Python. This allows students to use the same library API in the two languages. Holding the library constant and switching languages is thought to enable better migration between programming paradigms [3]. Scheme is a commonly mentioned uncommon language for computing, and there are some very good Schemes designed for use in education (e.g., Racket [4]). Calysto Scheme is a version of Scheme implemented in Python that uses all of Python's libraries [1]. Thus, Calysto Scheme has many of the advantageous that make Python popular, including being cross-platform, and providing an extensive library of functions. Processing is a language and environment that attempts to make Java as easy to use as Python, and is designed to allow the creation of beautiful and compelling art [6]. Processing allows some Java shorthand, so that the full Java language can be avoided, and many libraries are included by default.

Although all three of these uncommon languages are arguably better than Python, I argue that none will become common. In fact, I believe that the set "common languages for CS1" will remain small. This has little to do with the details of any particular language. Rather, "common languages" will comprise those that have the best support communities for teachers.

5. SCALA (MARK LEWIS)

Scala is a functional/OO hybrid language that runs on the JVM and is #14 on the RedMonk language ranking[5].

A. Scala is a rising language in industry, but does not have nearly the market penetration of Java, C++, or Python. Being based on the JVM does make the inevitable task of learning Java later easier.

B. Scala falls in a middle ground for resources. The educational resources are still thin, but there are well over a million questions that have been answered about the language on StackOverflow. Fortunately, it is easy to create problems for which no online solutions currently exist.

C. Scala is a statically-typed language with a uniform syntax, a REPL, and a scripting environment that encourages students to develop a Java-like coding style while also allowing coverage of functional concepts. Being a "real" language evolving to support professional developers does mean that it has complexities not found in teaching languages.

D. Unlike many languages, Scala is very good at both programming-in-the-small and programming-in-the-large. The main professional uses cases are Web Development, Big Data Processing, and any application that needs to scale up, so it can work well with later curricular elements. We adopted it at Trinity because it was good at covering all the concepts we wanted for CS1 and CS2.

6. REFERENCES

[1] Calysto Scheme. github.com/Calysto/calysto_scheme, 2015. [Online; accessed 28-August-2008].

[2] A. P. Black, K. B. Bruce, M. Homer, J. Noble, A. Ruskin, and R. Yannow. Seeking grace: A new object-oriented language for novices. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 129–134, New York, NY, USA, 2013. ACM.

[3] D. Blank, J. S. Kay, J. B. Marshall, K. O'Hara, and M. Russo. Calico: A multi-programming-language, multi-context framework designed for computer science education. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 63–68, New York, NY, USA, 2012. ACM.

[4] R. B. Findler, J. Clements, C. Flanagan, M. Flatt, S. Krishnamurthi, P. Steckler, and M. Felleisen. Drscheme: A programming environment for scheme. *J. Funct. Program.*, 12(2):159–182, Mar. 2002.

[5] M. Odersky, L. Spoon, and B. Venners. *Programming in Scala, 2nd Edition.* Artima Inc, 2011.

[6] C. Reas and B. Fry. *Getting Started with Processing.* Make Books - Imprint of: O'Reilly Media, Sebastopol, CA, 1st edition, 2010.