SICP JS: Ketchup on Caviar?

Martin Henz
National University of Singapore
henz@comp.nus.edu.sg

Tobias Wrigstad
Uppsala University
tobias.wrigstad@it.uu.se

With its minimalism, the language Scheme is well suited, if not designed, for teaching the structure and interpretation of computer programs (SICP) to freshmen computer science students, and Harold Abelson and Gerald Jay Sussman made use of the language in their eponymous book, whose second edition was published in 1996. The presenters applied the same minimalism to JavaScript, by identifying four sublanguages just expressive enough for the first four chapters of SICP, and named the languages Source §1, 2, 3 and 4. (There turned out to be no need for a sublanguage for chapter 5 of SICP.) Due to changes introduced to JavaScript with ECMAScript 2015, the Source languages are similar enough to Scheme for a relatively close adaptation of SICP to JavaScript. The resulting book by Abelson and Sussman as original authors, and by the presenters as adapters, is available online, including a side-by-side comparison.

We encountered the following issues during the adaptation due to the differences between the Source languages and Scheme, and briefly sketch here how they are resolved in SICP JS.

The distinction between statements and expressions, and the use of `return` is probably the most significant change from SICP to SICP JS. A notable consequence is the need to wrap return values in data structures in 4.1.1 and 4.1.3 in order to distinguish `x => { return x; }` from `x => { x; }` the latter of which returns undefined in JavaScript. We faithfully implement JavaScript's return statements in chapters 4 and 5, such that control can return to the caller from anywhere in the function body. This leads to several significant changes in these chapters, compared to the original. As a benefit, SICP JS helps readers understand statement-oriented languages such as Java and Python better.

Both Scheme and JavaScript (in strict mode, introduced in ECMAScript 5) employ lexical scoping. The Source languages only use JavaScript's const and let (introduced in ECMAScript 2015) and avoid JavaScript's var. The treatment of the scope of variables in chapter 4 and 5 becomes more uniform in SICP JS compared to SICP, as a result of consistently applying a treatment of const and let akin to Scheme's derived expression letrec.

The absence of Scheme's homoiconicity might at the surface be considered a major obstacle to adapting SICP to languages with a conventional syntax. However, SICP already hides the concrete syntax of programs behind an abstraction layer, which greatly simplifies the JavaScript adaptation. The introduction of an explicit parser suffices for adapting chapter 4 (including section 4.4 on logic programming), and the controller instructions in chapter 5 of SICP JS enjoy a syntax similar to SICP, through the use of constructors, which fit naturally into section 5.2.3. On the negative side, the lack of macros and our restriction to a JavaScript-compatible parser required significant changes to and occasionally replacement of exercises in chapter 4.

The audience is welcome to inspect SICP JS by visiting https://source-academy.github.io/sicp. A comparison edition lets the reader inspect the changes and compare them line-by-line with the original. The presentation will leave ample time for discussion.

The presentation will also cover the Source Academy, an online learning environment for programming, developed by and for students at the National University of Singapore, which implements the four Source languages along with several variants and extensions introduced in SICP.