Notes on Abelson and Sussman, *Structure and Interpretation of Computer Programs*

Chapter 1, "Building Abstractions with Procedures"

Cameron Clarke

July 2019

---

1. Building Abstractions with Procedures

- Target of study: **computational processes** (1)
  - intuition: "Computational processes are abstract beings that inhabit computers" (1)
- Processes manipulate **data** (1)
- Evolution of a process is directed by a pattern of rules called a **program** (2)
- An upshot: "**People create programs to direct processes**" (2)
- Programs are composed from **programming languages** (2)
- A framing thought, among others (2-3)

  > Well-designed computational systems, like well-designed automobiles or nuclear reactors, are designed in a modular manner, so that parts can be constructed, replaced, and debugged separately.

Programming in Lisp

- "[...] our **procedural** thoughts will be expressed in Lisp" (3)
- Lisp **interpreter**: "a machine that carries out the processes described in the Lisp language" (3)
- Interesting historical note, about one of Lisp's original use cases (4)

  > Lisp [...] was designed to provide symbol-manipulating capabilities for attacking programming problems such as the symbolic differentiation and integration of algebraic expressions.

- A note about dialects of Lisp (4)

  > Lisp is by now a family of dialects, which, while sharing most of the original features, may differ from one another in significant ways

  - This book makes use of the Scheme dialect of Lisp
- A main reason why Lisp is being used as the framework for the book's discussion of programming (5)

If Lisp is not a mainstream language, why are we using it as the framework for our discussion of programming? Because the language possesses unique features that make it an excellent medium for studying important programming constructs and data structures for relating them to the linguistic features that support them. **The most significant of these features is the fact that Lisp descriptions of processes, called *procedures*, can themselves be represented and manipulated as Lisp data**. The importance of this is that there are powerful program-design techniques that rely on the ability to **blur the traditional distinction between "passive" data and "active" processes**.

- The above facts make Lisp bood at writing programs that **manipulate other programs as data** (like interpreters and compilers)

1.1 The Elements of Programming

1.2 Procedures and the Processes They Generate

1.3 Formulating Abstractions with Higher-Order Procedures