1 Why I Still Read SICP

Every few years I return to *Structure and Interpretation of Computer Programs*. Not because I need the refresher, the ideas are by now old friends, but because the book is the clearest account I have found of what programming *actually is*.

Most programming books teach you a language or a framework. SICP teaches you that a program is a description of a process, and that understanding how processes evolve over time is the central skill of the discipline.[1]

1.1 The Language Is Not the Point

The book uses Scheme (a small, elegant Lisp) as its notation. Some readers get stuck on this. They want examples in Python or JavaScript, as though the language were the subject matter.

But the language is a *tool for thinking*. Scheme's parentheses enforce a discipline: there is no ambiguity in

```
(define (square x) (* x x))
```

compared to the many ways other languages let you write the same thing. The surface area of the language is small enough that it never gets between you and the idea.

1.2 Lisp Is Where the Modern Came From

It is easy to miss this if you learned programming after 2000. By then, the ideas Lisp introduced had been absorbed so thoroughly into mainstream languages that their origin became invisible.

Garbage collection. Lisp, 1959. The REPL. Lisp, 1964. First-class functions, closures, higher-order procedures, dynamic typing, homoiconicity, macros that operate on code as data. All of it comes from the same *House of Thought*, the one Abelson and Sussman are teaching in this book.

When Python added lambda. When JavaScript shipped closures. When Java eventually got generics and then streams. When Rust introduced iterators as first-class combinators. Each of those was a language catching up to ideas that were already mature and battle-tested in the Lisp world. The genealogy is not a matter of opinion. It is traceable, commit by commit, paper by paper, decade by decade, back to McCarthy's 1960 paper and to the interpreter that could be written in its own notation.

SICP is not a book about Scheme. It is a book about computation, and it uses Scheme because Scheme is small enough to get out of the way. What you are reading, underneath the parentheses, is the shape of the thing itself. The lambda calculus made executable, the Church encoding made concrete, the substitution model turned into something you can run. The lambda is not decoration. It is the whole story.

---

[1] Abelson and Sussman open with: "We are about to study the idea of a *computational process*. Computational processes are abstract beings that inhabit computers." The word "inhabit" is doing real work.

1.3 The Base Is Everything

There is a temptation, when learning to program, to stay close to the surface. To learn the framework before the language. To learn the library before the algorithm. To learn the tool before the principle. This works, in the short term. You can build a great deal on a foundation you do not fully understand.

Until you cannot.

The moment you hit a performance cliff you cannot explain, a concurrency bug you cannot reason about, an abstraction that does not compose... that is the moment the foundation matters. And if you never built it, you have no ground to stand on.

This is not only true of programming. The base is everything in life. The musician who cannot hear intervals is trapped by sheet music. The carpenter who does not understand wood grain is at the mercy of the material. The programmer who has never written an interpreter does not really know what a variable is but only what it does in the particular system they learned first.

SICP is one of the few books that builds the base deliberately. It does not give you Scheme and then teach you to program. It builds an interpreter, then builds abstraction on top of the interpreter, then shows you how the interpreter could have been different and what that difference means. By the end you have not just learned to program. You understand what programming is built on and that understanding does not go stale.

That is why I keep coming back. Not for the exercises, not for the Scheme, not for the nostalgia. Because every time I return I find something in the foundation I had not seen clearly before, and it changes how I see everything above it.