1) Students will learn to program, using a dialect of the Scheme programming language. That is, given an abstract problem description (or specification), students should know how to choose appropriate data structures and algorithms, design the appropriate functions and how they will fit together (i.e., functional decomposition), incrementally develop their code using debugging techniques, and verify their code works using a testing framework. The inputs and outputs of their program are (at the very least) Scheme expressions.

2) Students will be able to generate and debug recursive programming techniques, including linear recursion and advanced “tree” and “mutual” recursion.

3) Students will be able to generate and debug “Higher Order Function” (HOF) (aka “functional” in CS3S) programming techniques. This means functions may themselves be used as input or output to other functions. Students are familiar enough with the concept of “lambdas” (unnamed functions) that they can use them in HOF code.

4) Students are to gain some appreciation of the algorithms underlying tools like databases, spreadsheets, and expert systems.

5) In the process of learning how to program, they will be applying knowledge of mathematics, science, and engineering.

6) Students will work on a (often two-person) final project, requiring a fair bit of software engineering. They will be making design decisions (algorithm, data structure, representation, etc.) and considering various implementation trade-offs. They will often interface with an existing code base or library.

Note: CS3L and CS3S are two nearly identical courses, with the same course title: “Introduction to Symbolic Programming”. They differ in very subtle ways (one uses the Harvey Simply Scheme book, and one uses the Grillmeyer Exploring Computer Science with Scheme book), and format (one is self-paced and one is lab-centric), but both should have the same outcomes list. I indicate where they differ in their summaries.