Learn Programming Like It's 1986

Syllabus

Logistics

The class is mostly just the lectures, though we throw in a bit of problem-solving to make sure you get to practice the concepts. We will provide lecture notes and point you to reading sources if you have to miss a class, but since this class moves very quickly, please try your best to attend *all* the lectures!

Laptop is not required but recommended! Scheme is designed to be very blackboard-friendly, so most of the teaching will be done on the blackboards, with occasional computer demonstrations mixed in. You might find it helpful to follow along by typing the code on your laptop, though, since that will help you get the feeling and build some muscle memory.

Setting the expectations

Since the class is six weeks long and there is no mandatory homework, you likely will *not* become proficient in programming—after all, it takes a lot of practice. The class will, however, allow you to appreciate the beauty in some of the ways of thinking that computer scientists do. You will be more than well-equipped to take more hands-on courses (like Harvard's freely available CS50 and MIT's 6.100A) and flourish in whatever you choose to do, whether that would be game programming or web development, thanks to your ability to handle complex abstractions in programming.

There are supplementary readings and exercises available for students who want to get more hands on during this class.

Contents

Week 1: Scheme and black-box abstractions
We briefly explain the history of MIT's introductory CS curriculum, specifically 6.001 and SICP. We walk through how to write programs in Scheme, using a simple IDE called DrRacket. We learn how to express procedures with conditionals and recursion. In the process, we explore a simple yet powerful algorithm for estimating square roots as a cool bonus. We might also build an efficient algorithm for the classic number guessing game if time permits.

Week 2: Shapes of processes
We expand upon the ideas from week 1 by going through Scheme's evaluation rules more carefully. We learn how small differences in our procedure descriptions give birth to processes with vastly different shapes. We introduce the notion of recursive and iterative processes then practice building a lot of those processes. Some cool examples include fractals and the classic Tower of Hanoi.

Week 3: Higher-order procedures and data abstraction
We learn about higher-order procedures, procedures that can take other procedures as input and emit procedures as output! You might get a sneak peek into the heart of calculus too. After that, we learn to construct and manipulate compound data, starting from basic examples like rational numbers and vectors. If time permits, we will also lay the groundwork for the Turtle graphics language which should be finished in the next week.

Week 4: Data abstractions continued
We continue directly from week 3, generalizing the notion of data abstraction so we can deal with arbitrarily large lists of data. Using this, we can fully design and implement our Turtle graphics language! We will also walk through some classic algorithmic questions pertaining to linked lists.

Week 5: Mutability
The real world is all about *change*. This week, we depart from the world of functional programming and introduce the concept of mutable states. With states, we can build more efficient and interactive programs! Of course, our functional programming model will still prove incredibly helpful in understanding how all of this works.

Week 6: Advanced topics
Depending on class interest, we could go into any of these topics:

- Hierarchical data structures (Binary search trees, etc.)
- Metacircular Evaluator (Writing Scheme programs to interpret Scheme programs to interpret Scheme programs to interpret Scheme programs to…)
- Survey of other programming languages (Python, SQL, Assembly)

We might dive more deeply into previous weeks' contents (just in case we couldn't cover everything in time). We'll make sure to include recommendations on what to study afterward if you would like to continue learning about programming!