Many of you may be like me and came into this class with only an imperative background in programming languages. Getting my mind to think less about black-box connections and more about the fundamentals of abstraction has been a challenging exercise, especially as we begin to dive into higher order process construction. Like I am sure a lot of you did, I began googling about Lisp, Scheme, SICP, etc... So, I was hoping to compile a collection of available online resources for understanding this class to help everyone be better prepared going into lecture. In addition, I thought it might be helpful to understand the context by which we are learning something like SICP historically (I require such a tether). Here is what I have found so far:

SICP Digital Copy
PDF version of the textbook for all your CTRL+F needs

SICP Lectures (1986)
A lecture series given to HP employees in the late 80s by Abelson and Sussman covering the material of SICP. These lectures have been especially helpful as a supplement to the Q/A style of 403/503. If you watch nothing else, watch the introduction. It really does well to frame the discussion for Scheme and the class. Course is served by MIT OpenCourseWare.

MIT iCampus Online Tutor
This is an online reference for students who took the old MIT class. It contains slides, resources and problem sets for SICP. You do have to register for this.

Hacker News Discussion
Of an article discussing why SICP/Lisp was ditched at MIT in favor of "programming by poking" using Python. TLDR; it was because Python was more "practical" for today's process of engineering, but that doesn't mean there isn't a large group who fundamentally disagrees with the decision. Here are some nuggets from the HN thread:

u/sitkack
Programming by poking that I see folks doing, isn't science. It is stabbing in the dark until something appears to work. There is more to computational thinking than throwing some APIs together in a blender and making a cool demo.

u/Const-me
Looks like another face of a centuries-old debate between theoretical and experimental science. The theoretical scientists play with abstractions they fully understand. Experimental scientists poke at things they don't fully understand. They tend not to do well with each other. When computers were in the infancy, they were viewed primarily as a scientific tools to crunch numbers and play with abstraction. That's the theory-oriented view that I think is shared by the SICP authors.

Then computers became complex, fast, and capable of doing much more than crunching numbers and evaluating those S-expressions. While you can view a modern web browser, or MS Word, or a videogame, as a program for a Turing complete machine, this does not mean you should. Too little value in that: sure there are instructions and state, but now what? More experiment-oriented
mindset usually works better with those kind of problems.

Further Explanation of MIT Decision (7min) by Gerald Sussman in which he explains that SICP is meant to provide a "language of engineering" as much as a course in computer science.

I will be updating the post if I come across anything new. Feel free to post in your own links. I just felt this was really helpful for me to understand the context of what we are learning, so I thought I would share!

Subject: Re: Understanding SICP
Posted by lusth on Fri, 02 Sep 2016 18:37:23 GMT

I liked this quote:

I self-taught myself, using "programming by poking". and I wrote garbage code.

Then I got a CS degree, and learned proper structure and organization of code.
It's bad enough now, when a large contingent of code is produced by non-educated "pokers".

I weep for the future where ALL code is produced by "pokers".