# Foreword

This booklet is based on a challenge Will E. Byrd set himself in January 2024: to overcome procrastination by committing to writing and publishing 11 books and 1000 YouTube videos during the calendar year.

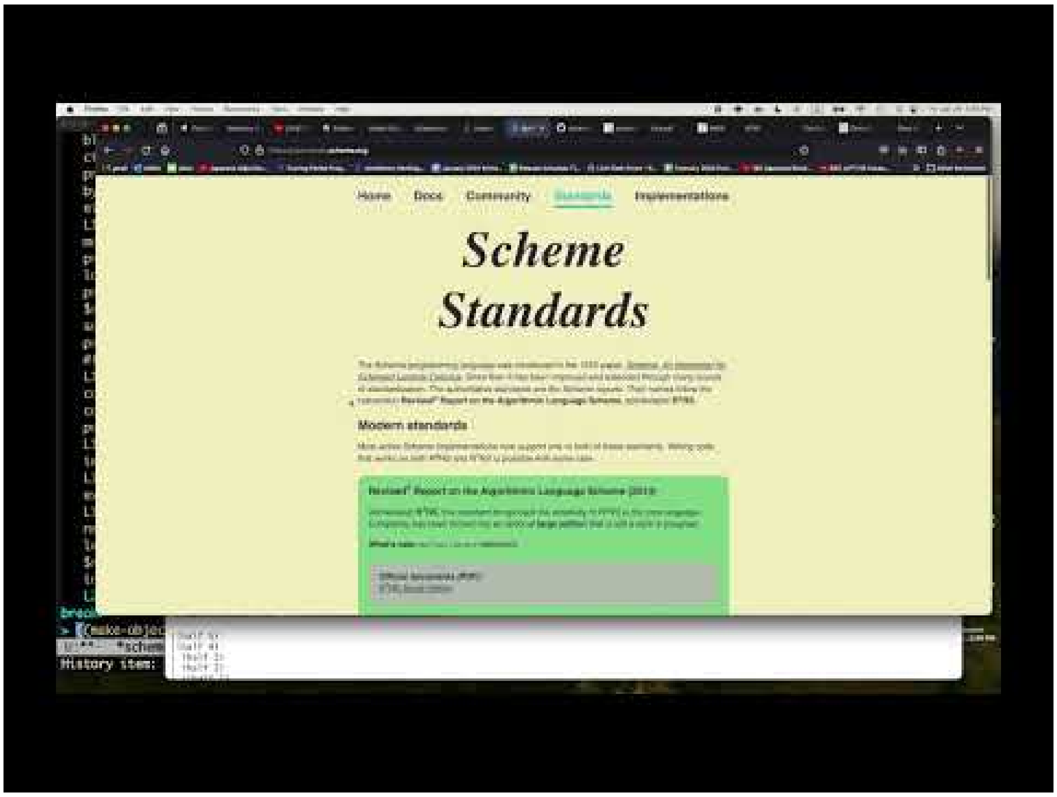
# Shownotes

## January 2024

### January 26th - Scheme Learning Resources

In which Will Byrd shows us some starting points for diving into his favourite programming language

# January 26th 2024: Scheme Learning Resources



Watch Will's video at https://youtu.be/iC8eSdoyu9A

**Words by Will E. Byrd**

The following is a (very) lightly edited version of the transcript of Will's video linked above. Links to content mentioned have been located and included for further reading/viewing.

## Intro

Welcome to Episode IV of **Will's Guide**. This year, I've decided to make 1,024 videos - I've upped the number from a thousand, because I think two to the tenth or one kiloTube of videos is more fun!

I've also set up the Discord server, and we've had about a dozen people join. If you're interested in joining the Discord server *The Imperishable Wonderland of Infinite Fun*, please email me.

Today, I'd like to talk about something close to my heart: Scheme, a language which I love. And I'm just going to talk a little bit about how you might learn Scheme. I'm going to have a whole bunch of videos on Scheme (I hope), but how do you learn about Scheme? What are the resources available? Let's just talk about a few of them I found useful..

### *The Little Schemer* by Dan Friedman and Matthias Felleisen

- The Little Schemer (MIT Press)
- Dan's Wikipedia Entry
- Felleisen.org Site, Matthias' Wikipedia and Racket.org

One is called *The Little Schemer* by Daniel P Friedman and Matthias Felleisen. Dan was my adviser at Indiana University for my PhD, and he was also Matthias's adviser. You also might know Matthias from the Racket world, so you have two experts guiding you.

This book is really about thinking computationally, and specifically thinking about *recursion* - recursion over lists and trees - those sorts of things, and also a little interpreter work. So, if you think is is going to be a book that's a big thick manual on Scheme - the programming language and the semantics or whatever - well, you might be disappointed. But this really in another way is at the heart of thinking about Scheme computationally and thinking about computation and recursion, so it's a great book to affect your thinking and make sure that you're really comfortable with ideas about recursion.

So this is recommended by me if you want to learn how to think recursively at least a certain way of thinking recursively. If you're not comfortable with recursion, if you read this book and if you learn Scheme in general, you're likely to become very comfortable with recursion, because Scheme is really based on recursion. There's no loop constructs like there are in most languages (or anything does look like a loop, it's actually tail recursion which we'll talk about later)..

So it's sort of the secret that Scheme doesn't really have loops in the way most languages do, although it turns out you can implement loops that are equivalent to a certain type of recursion..

## The Structure and Interpretation of Computer Programs by Hal Abelson, Jerry and Julie Sussman

- The Structure and Interpretation of Computer Programs

Another great resource is *The Structure and Interpretation of Computer Programs* by Harold Abelson and Gerald J Sussman with Julie Sussman. This is very famous as maybe the greatest textbook in the history of computer science - it was used as the introductory textbook at MIT for many years. It has all sorts of interesting ideas about computation and interpreters - how to write interpreters, how you write a program that interprets another program, those sorts of ideas, computational models. Great book, full of deep fascinating ideas.

Once again not a manual on Scheme - Scheme is introduced as needed as a computational notation which really gets back to the heart of Lisp, the Lisp family of languages. You need to have some sort of notation to talk about computation, so let's have a very elegant, simple, minimal notation that's compositional, has nice properties, and we can think about that's at the heart of Scheme, and that's one reason I really like Scheme.

## MIT 6.001 1986 Lecture Series

- MIT 6.001 1986 Lectures on YouTube

There's a series of videos based on the ideas in the books called the *SICP Lectures* and if you're interested in the book you might like these videos.

I will say that the videos can be - and the book can be - a little mathy, especially in terms of the examples. They aim towards the MIT entering class, or in this case, for these lectures I don't know if these are supposed to be more experienced programmers or whatever, but it can turn people off a little bit. So if you find that this is a little hard going, maybe start with something like *The Little Schemer* first, which is really written for the bright high school student, but the ideas are very deep and powerful, and can benefit anyone. So maybe start with something like *The Little Schemer*, and if you feel that's not enough, you could try *The Reasoned Schemer* which is a much harder book, but in any case you could build yourself up towards the ideas in SICP over time.

## Software Design for Flexibility by Chris Hanson and Jay Sussman

- Software Design for Flexibility

There's a follow-on book - intellectually a follow-on book - called *Software Design for Flexibility*: *How to Avoid Programming Yourself into a Corner* by Chris Hanson and Gerald J Sussman. this is metaphorically the advanced version or the advanced follow-up of *Structure and Interpretation of Computer Programs*. It's really about how do you write software, how do you write complicated programs in such a way, that if you want to make a significant change to the program, you don't have to throw away all your code and start over again, which turns out to be quite tricky. So the book uses a bunch of techniques people in the Scheme and Lisp community, and AI communities, have developed over a long period of time to make software more flexible.

OK, so it's full of really interesting techniques should probably talk about that at some point. Once again, not a book on Scheme but it contains a lot of really advanced uses of Scheme - you can learn a lot by looking at this book.

## Simply Scheme: Introducing Computer Science by Brian Harvey and Matthew Wright

- Simply Scheme: Introducing Computer Science

If you want something maybe closer to a traditional textbook there's *Simply Scheme* by Brian Harvey and Matthew Wright. Now you can see the second edition is from 1999, so it's, you know, a quarter of a century old - Scheme has changed and so forth, but the basic ideas will be similar.

So there there are some other introductory textbooks on Scheme. You'll find most of them are kind of out of date in terms of the specific language features, but the important thing is how you think about the language. You can learn about the new variants of Scheme over time.

## Scheme and Functional Programming Workshop

- Scheme and Functional Programming Workshop

There are also a bunch of papers that you can you can find on Scheme. There's a *Scheme and Functional Programming Workshop*, it's been held for a number of years, since 2000. Hopefully there'll be a 2024 one happening soon. I'm on the Steering Committee for this Workshop and I've organized it several times, so if you look at one of these workshops like from last year, you can find the Program and the Accepted Papers, and this sort of thing. So here's a pre-print and you find this is a recorded talk, on "Designing a Language for Learning Continuations".

You can find the video online - lots of good resources there. This is going to be more advanced, of course, than a textbook, but this is where a lot of the ideas get percolated - or more like disseminated - through the community, is through papers or researchers talking to each other.

If you don't have access to researchers talking to each other, if you're not in grad school, if you're not a working academic, if you're not a Scheme implementer, then reading papers is probably the best thing to do.

## Lambda the Ultimate

- Lambda the Ultimate (papers page)

Certainly one of the best things to do now, because of the internet, and the web, there are lots and lots of videos you can watch and learn things that way as well. And of course, communicate with people via forums and all the amazing things that we have now. Speaking of forums, you know there's *Lambda the Ultimate*, the Programming Languages Web Log, which used to be more active. I don't know to what extent it is active these days, but there's a period in time where this was a really important resource for people trying to understand functional programming. And there's a group of papers here called Lambda the Ultimate papers, where the website got its name, and you see *Lambda the Ultimate Imperative*, for example (Uh, I don't know what will happen if I try opening that, let's see how about that one yeah let's try it - I have no idea - oh, okay)..

Well, anyway these are papers with Guy Steele and Jerry Sussman, these the original Lambda the Ultimate papers: Lambda the Ultimate "X". These are papers that were describing Scheme and using Scheme or using Lisp to solve different types of problems and looking at Lips or Scheme from different perspectives, so these are classic papers.

"Introduction to Lambda Calculus," ok, "GOTO Statement Considered Harmful" by Dykstra, a whole bunch of interesting papers, "Essence of Compiling Continuations," so, you know, these are classic papers in functional programming, in programming languages, that are worth taking a look at, and many of these are involving Scheme or some version of Lisp.

## Scheme.org

- Scheme.org

## SchemeDoc Bibliography

- SchemeDoc Bibliography

## R5RS Page

- R5RS Page
- R5RS Standard PDF

## R6RS

- R6RS.org

## R7RS

- R7RS.org

## The Scheme Programming Language by R. Kent Dybvig

- The Scheme Programming Language

## The Chez Scheme User's Guide

- The Chez Scheme User's Guide

## Chez Scheme on Github

- Chez Scheme on Github
- Chez Scheme Github Repo

## Scheme Related Papers by CollectRobot

- Scheme Related Papers by CollectRobot
- How to Debug Scheme Programs
- Generation-Friendly Eq Hash Tables

# Appendix A - Dev Notes for this Booklet Project

To upload my repo to Github, I ran:

```
$ gh repo create
```

To create a `gh-pages` branch in Github, I followed this advice:

```
git checkout --orphan gh-pages
git reset --hard
git commit --allow-empty -m "Initializing gh-pages branch"
git push origin gh-pages
git checkout main # was master
```

I added entries to the Makefile provided here

Now, I run

```
git commit -a -m "publish"
git push
make deploy
```

# Index