

SICP in Emacs

Konstantinos Chousos

June 7, 2023

Table of Contents

1	Getting the book	2
1.1	Obtaining the info file	2
2	Setting up Scheme	3
2.1	Racket in Emacs	4
2.2	Racket in Org-Babel	4
3	Result	5
4	Miscellaneous tips	6

I recently began reading the notorious “Structure and Interpretation of Computer Programs” [1], a.k.a. the *Wizard book*. I’m only on the first chapter, but I can already see its value and why it gets recommended so much.

From Wikipedia:

Structure and Interpretation of Computer Programs (SICP) is a computer science textbook by Massachusetts Institute of Technology professors Harold Abelson and Gerald Jay Sussman with Julie Sussman. [...] It teaches fundamental principles of computer programming, including recursion, abstraction, modularity, and programming language design and implementation. [...] The book describes computer science concepts using Scheme, a dialect of Lisp. It also uses a virtual register machine and assembler to implement Lisp interpreters and compilers.

In this post, I aim to showcase my workflow for studying the book using Emacs [2]. Also, I will provide any resources that helped me get going. To study SICP, we need two things: The book and a Scheme implementation for the examples and exercises.

1 Getting the book

Lucky for us, the book is freely distributed from MIT itself. It is available in [HTML](#) and [PDF](#). But, there is also a third format option and it is the one we're going to choose: the Texinfo format.

From the official GNU site¹:

Texinfo uses a single source file to produce output in a number of formats, both online and printed (HTML, PDF, DVI, Info, DocBook, LaTeX, EPUB 3). This means that instead of writing different documents for online information and another for a printed manual, you need write only one document.

The Texinfo system is well-integrated with GNU Emacs.

That last line is what's important here. info files are essentially manuals in plain text. Emacs has a built-in mode for rendering such documents. By using the info format, we can read SICP from inside Emacs.

1.1 Obtaining the info file

The info file can be retrieved in two methods:

1. By installing the sicp MELPA package²
2. By downloading the info file directly from [neilvandyke.org](#) and installing it.
 1. Download the `sicp.info.gz` file ([link](#)) in your home directory.
 2. Execute the following commands

```
1 $ sudo cp ~/sicp.info.gz /usr/local/share/info/
2 $ sudo chmod 644 /usr/local/share/info/sicp.info.gz
3 $ sudo install-info /usr/local/share/info/sicp.info.gz
   ↳ /usr/local/share/info/dir
```

Now SICP will be available through Emacs! To access it, you need to open Emacs, type `C-h i` to go to the `*info*` top directory, type `m` to search and type `sicp` to find the book. If everything went correctly, you should be greeted with something like this:

¹<https://www.gnu.org/software/texinfo/>

²<https://melpa.org/#/sicp>

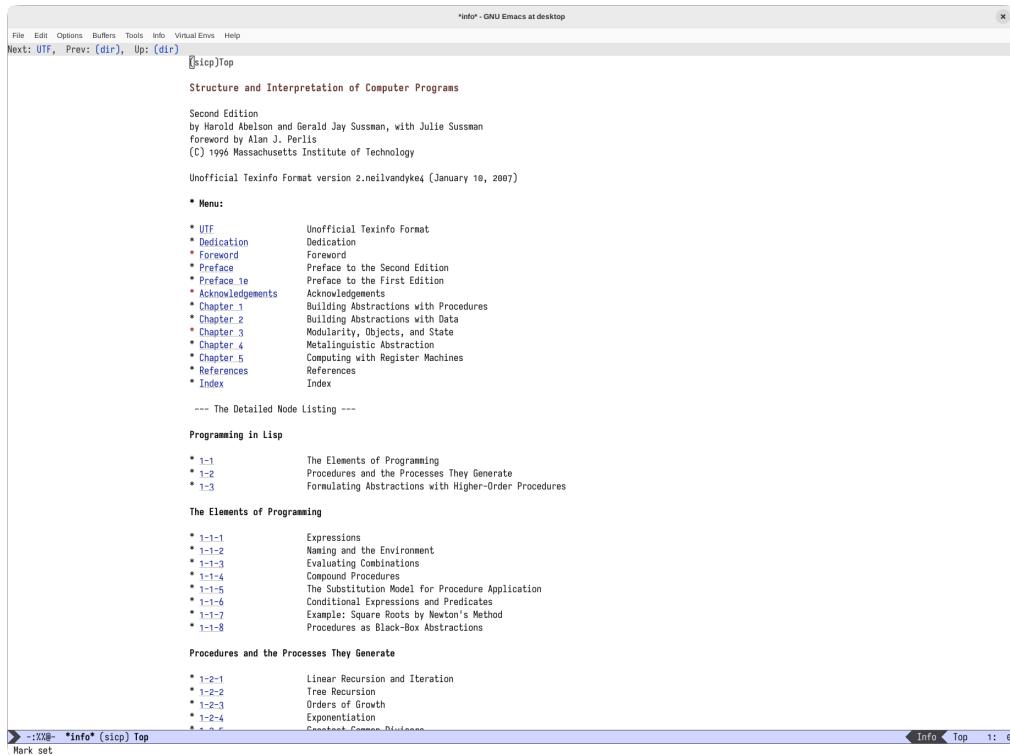


Figure 1: SICP’s table of contents in ‘info’ format, viewed from within Emacs

2 Setting up Scheme

SICP’s examples and exercises are all implemented in Scheme. Scheme is a Lisp dialect with many implementations. SICP uses the [MIT-Scheme implementation](#). Turns out GNU/MIT-Scheme is **not** fully compatible with the code in SICP ([source](#)). Instead, we will use [Racket](#). Racket offers the [SICP collection](#), a Racket #lang that makes it fully compatible with the SICP code.

First, we need to install racket through our package manager. After that, the sicp collection can be downloaded like this:

```
1 $ raco pkg install sicp
```

That’s it! Now, when we write a .rkt file that needs to be compatible with SICP all we need to do is add `#lang sicp` at the top of the file³.

³when using the REPL, we need to first evaluate `(require sicp)` before evaluating anything else.

2.1 Racket in Emacs

Personally, I recommend [racket-mode](#) for working with Racket in Emacs. Another popular choice is [geiser](#), but its geiser-racket module seems to be unmaintained⁴.

To install racket-mode using [elpaca](#), add the following to your config file:

```
1 (use-package racket-mode  
2   :elpaca t)
```

2.2 Racket in Org-Babel

In case you choose to go the literate programming route (as I have) using Org-Mode, you will need to enable support for racket in org-babel. To do this, use the [emacs-ob-racket](#) package. Add the following to your config:

```
1 (use-package ob-racket  
2   :elpaca (:type git :host github :repo "hasu/emacs-ob-racket"))
```

and then enable racket in your org-babel configuration.

```
1 (org-babel-do-load-languages  
2  'org-babel-load-languages  
3  '((emacs-lisp :tangle ./init.el . t)  
4    (C . t)  
5    (python . t)  
6    ...  
7    (racket . t)))
```

To be able to use the sicp package in org-babel code blocks, you need to add :lang sicp in the Org block, like so:

```
1 #+begin_src racket :lang sicp  
2 "Hello World!"  
3 #+end_src
```

Instead of adding that to every code block, you can add `#+property: header-args :lang sicp` to the start of your Org file. This will be applied to *all* code blocks in the file, so make sure you include only racket code blocks. This can be mitigated by specifying that these header-args are to be applied only to racket blocks, like so: `#+property: header-args:racket :lang sicp`.

⁴<https://lists.nongnu.org/archive/html/geiser-users/2022-06/msg00004.html>

3 Result

After all this work, now we can finally start reading SICP. My so-far workflow consists of the book in the left window, a racket REPL in the top-right corner and my Org-Roam notes in the bottom-right corner.

The screenshot shows a desktop environment with three windows open:

- Racket REPL (top-right):** A terminal window titled "Racket REPL ->". It contains Scheme code related to the exercise 1.1.5 from SICP. The code includes definitions of functions like `car`, `cadr`, `cond`, `if`, and `map`, along with some arithmetic operations and logical expressions.
- Book (top-left):** A window titled "SICP ->". It displays the first page of the book "Structure and Interpretation of Computer Programs" by Abelson, Harold and Sussman, Gerald Jay and Sussman, Julie. The page is titled "1.1.5" and discusses the sequence of expressions in Scheme.
- Org-Roam notes (bottom-right):** A window titled "Julie :: Structure and Interpretation of Computer Programs". It contains an Org-mode document with sections like "1.1.5", "Applicative order", "Evaluating nested expressions", and "Normal order". It also includes a code block for the `map` function.

Figure 2: My SICP studying workflow

When it comes to the exercises, I use Org-Mode and Org-Babel to write the solutions in a literate programming style. The file is divided by chapter. Each exercise is included followed by its (hopefully correct) solution. (So far) I use a single .org file and export it to PDF. Also, all of the code blocks are exported to a .rkt file, with links to the corresponding position in the org file. All of these files can be found at [this repo](#).

The screenshot shows a GNU Emacs window with a buffer titled 'sicp.pdf'. The buffer contains Scheme code for exercises from SICP. The code includes definitions of procedures like `(define a 3)`, `(define b (+ a 1))`, and `(+ a b (* a b))`. It also includes code for translating expressions into prefix form, such as `(+ 4 (* 2 (- 3 (- (+ 4 5) 3))))` which evaluates to `5`. Another part of the code defines a procedure `(proc a b c)` that takes three numbers and returns the sum of the squares of the two larger numbers. The results pane shows the output of these evaluations.

Figure 3: My SICP solutions in literate programming

4 Miscellaneous tips

- **Update 07/06/2023:** As [u/jherrlin](#) on Reddit pointed out, the fact that SICP is in text format gives us the ability to leverage Emacs' built-in bookmarks feature. When you arrive to the end of your study session, just type `C-x r m` and a bookmark will be placed on the current line. You can search your bookmarks with `C-x r b` or list them with `C-x r l`.

My tip is to name the bookmark the same each time (e.g. `sicp`). That way, when you re-create it in a later position, the old bookmark is discarded automatically. Also, if you run Emacs in daemon mode, I suggest to run `M-x bookmark-save` after adding a bookmark, to make sure it has been saved.

References

- [1] H. Abelson, G. J. Sussman, and J. Sussman, *Structure and Interpretation of Computer Programs* (Electrical Engineering and Computer Science Series), 2. ed., 7. [pr.] Cambridge, Mass.: MIT Press [u.a.], 2002, 657 pp., ISBN: 978-0-07-000484-9.
- [2] R. M. Stallman, “EMACS the extensible, customizable self-documenting display editor,” *ACM SIGPLAN Notices*, vol. 16, no. 6, pp. 147–156, Jun. 1981, ISSN: 0362-1340, 1558-1160, doi: [10.1145/872730.806466](https://doi.org/10.1145/872730.806466). [Online]. Available: <https://dl.acm.org/doi/10.1145/872730.806466>.