SICP in Emacs

Konstantinos Chousos

June 7, 2023

Table of Contents

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

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:

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

²https://melpa.org/#/sicp

		info - GNU Emacs at desktop				
File Edit Options Bulfers Tools Inho Vinual Envis Help						
Next: UTF, Prev: (dir), Up: (dir)						
[sicp] Top						
	Structure and Interpretation of Computer Program					
	Servere and inter	Scructure and interpretation of computer riograms				
	Second Edition	Second Edition				
	by Harold Abelson and Gerald Jay Sussman, with Julie Sussman					
	foreword by Alan J. Perlis					
	(C) 1996 Massachusetts Institute of Technology					
	Unofficial Texinfo Format version 2.neilvandyke4 (January 10, 2007)					
	* Nenu:					
	* IITE	Unofficial Taxinfo Format				
	* Dedication	Dedication				
	* Foreword	Foreword				
	Preface	Preface to the Second Edition				
	Preface 1e	Preface to the First Edition				
	Acknowledgements	Acknowledgements				
	Chapter 1	Building Abstractions with Procedures				
	• Chapter 2	Building Abstractions with Data				
	* Chapter 3	Notalianistic, Julietas, and State				
	* Chapter 5	Computer with Register Machines				
	 References 	References				
	* Index	Index				
	The Detailed Node Listing					
	Programming in Lisp					
	* 1-1	The Elements of Programming				
	* 1-2	Procedures and the Processes They Generate				
	* 1=3	Formulating Abstractions with Higher-Order Procedures				
	The Elements of Programming					
	• 1-1-1	Expressions				
	* 1-1-2	Naming and the Environment				
	* 1-1-3	Evaluating Combinations				
	1-1-4	Compound Procedures				
	1-1-5	The Substitution Model for Procedure Application				
	1-1-6	Conditional Expressions and Predicates				
	1-1-7	Example: Square woots by Newton's Method				
	1-1-0 LINGANIAP AP DISKLANK WAPPI ACTINUS					
	Procedures and the Processes They Generate					
	* 1-2-1	Linear Recursion and Iteration				
	1-2-2	Tree Recursion				
	1-2-3	Urders of Growth				
	1-2-4	Exponentiation				
-:XX@- *info* (sicp) Top		Info	Top 1:			
Mark set						

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:

\$ raco pkg install sicp

1

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:

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

1

2

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.

⁴https://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.

Macket HEP, etr - 6th Email at deltap		
File Edit Options Butters Tools Racket-REPL Vetual Envis Complete In/Dut Help		
Next: 1-1-7, Prev: 1-1-6, Up: 1-1 The value of a `not' expression is true when the expression <e> evaluates to faise, and faise otherwise.</e>	> (define a 3) > (define b (* a 1)) > (* a b (* a b))	
	19 > (- s b)	
Notice that 'and' and 'or' are special forms, not procedures, because the subexpressions are not necessarily all evaluated. 'Not' is an ordinary procedure.	#f > (if (and (> b m) (< b (* m m))) b a)	
As an example of how these are used, the condition that a number x be in the range 5 < x < 10 may be expressed as	$\stackrel{4}{\rightarrow} (cond ((= a, i, b)) ((= b, i), (= b, 7, a))) ((= b, i), (= b, 7, a))$	
(and (> x 5) (< x 10))	16) Contraction (Serv. 9)	
As another example, we can define a predicate to test whether one number is greater than or equal to another as	<pre>(er((-xy))(-xy))) >[</pre>	
(define (>= x y) (or (> x y) (= x y)))]		
or alternatively as		
(define (>= x γ) (not (< x γ)))		
"Garcis 1.1" Balow is a sequence of expressions. What is the result printed by the interpret in response to each expression? Assume that the sequence is to be evaluated in the order in which it is presented.	E ¹⁴ F Sector ID , OP ULD: Aberton, Hands and Susanae, Savid Jay and Susanae, Julis :: Structure and Interpretation of Comput ULD: Aberton, Hands and Susanae, Savid Jay and Susanae. Julis :: Structure and Interpretation of Comput	Rackat-REFI. Bot 28: 2 ter Programs
10	© 1.1.5	
(+ 5 3 4)	Υπάρχαυν δύο τρόποι για να γίνει evaluate μια έκφραση:	
(- 9 1)	- Applicative order	
(/ 6 2)	Είναι πιο αποτελεσματική γιατί οδηγεί σε λιγότερους υπολογισμούς	
(+ (* 2 4) (- 4 6))	- Normal order	
(define a 3)	Η normal σειρά αναλύει τους τελεστές μέχρι να φτάσει σε primitives, χωρίς να κάνεις κάτι evaluate. Μόν τελεστές είναι ορηπίτισε κάνει τους μπολογομορίς	νο όταν όλοι οι
(define b (+ a 1))	Για εφαρμογές όπου όλοι οι τελεστές είναι απλά "συντομογραφίες" το δύο μοντέλα βγάζουν ίδια αποτελέ	σματα (βλ.
(+ a b (* a b))	άσκηση 1-6).	
(= a b)	o cond	
(if (and (> b a) (< b (* a b))) b a)	ρεε rasket (cord (ΦΦ (Φ) (ΦΦ 42α) 	
[cond [[= a 4] 6]	(49b 49b))	
((= b 4) (+ 6 7 a)) (else 25))	• II	
-:::::	fo 55% 132: 28 Braindump Sitzmaster U: abelsonStructureInterpretationComputerInnez.org	Org Tcp 13: 0

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.

siqued* - GNU limaos at de	esitop x
 Edit Options Buffers Teels POF-Outline Image Visual Envis PDFTcols Help 	
$\int_{1}^{1} \int_{1}^{1} \int_{1$	<code-block><code-block><code-block><code-block><code-block><code-block><code-block><code-block><code-block><code-block><text><text><text><text><text><text></text></text></text></text></text></text></code-block></code-block></code-block></code-block></code-block></code-block></code-block></code-block></code-block></code-block>
AP bitimister -:e- sicplong Org 63X 118: e SICP]	All 1:

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

- 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. [Online]. Available: https://dl.acm.org/doi/10.1145 /872730.806466.