"Common Lisp Recipes" (ISBN 978-1-4842-1177-9)
Errata and Addenda

The newest version of this document is always available at http://weitz.de/cl-recipes/; you'll also find updated code downloads there.

Please report any errors not already listed here to edi@weitz.de.

- The MOBI version seems to deviate from the printed book and the PDF version (which to my knowledge have exactly the same content). For example, footnotes 30 and 31 in chapter 1 contain slashes in places where backslashes should have been used.
- **Page XXI**: "book" (end of second paragraph) should have been "books."
- **Page XXVII**: "execpt" should have been "except."
- **Page 2**: The definition of the `P1` package should better look like this:

```
(defpackage :p1
  (:intern :alpha)
  (:use :cl)                                                    ;; <-- added
  (:export :bravo :charlie))
```

  The reason is that the list of packages that `P1` will inherit from if `:USE` is not explicitly specified and thus implementation-defined. It will include (but will not necessarily be limited to) the `COMMON-LISP` package in many Lisps, but not in all. For example, SBCL and ABCL won't automatically inherit from `COMMON-LISP`.

- **Pages 2 and 117**: *Lisp* is not short for "**LIS**t **P**rocessing," but rather for "**LIS**t **P**rocessor." See http://www-formal.stanford.edu/jmc/recursive.html.
- **Page 16**: Should be `#p"/tmp/nanook.jpg"` instead of `#"/tmp/nanook.jpg"`
- **Page 20**: In the first box it should be:

```
CL-USER > (defpackage :quux (:use :cl))
#<The QUUX package, 0/16 internal, 0/16 external>
CL-USER > (in-package :quux)
#<The QUUX package, 0/16 internal, 0/16 external>
QUUX > (loop for s being each present-symbol collect s)
(COLLECT PRESENT-SYMBOL S BEING EACH FOR)                    ;; <-- changed
QUUX > (loop for s being each symbol of (find-package "QUUX")
             count t)
985
```

  (The difference is that the book claims you should also see the symbol `OF` in the line marked with a comment.)

- **Page 28**: "succint" should have been "succinct."
- **Page 35**: "the only compound objects in Lisp where conses" should have been "[...] were conses," of course.
- **Page 38**: Here's a better alternative to the "elegant" version of `INTEGER-TO-BIT-LIST`:

```
(defun integer-to-bit-list (x)
  (check-type x (integer 0 *))
  (reverse (map 'list 'digit-char-p (write-to-string x :base 2))))
```

  Note how `DIGIT-CHAR-P` (look it up in the HyperSpec) is a *predicate* which uses the COMMON LISP concept of a *generalized boolean* in a helpful and clever way: if its argument, a

character, does not designate a digit, the return value is NIL, but if it *does* stand for a digit, the return value is not just T but the digit's numerical value.[1]

- **Page 40**: Note that transposing a matrix this way might fail for larger matrices if the value of CALL-ARGUMENTS-LIMIT is too small. It can be as small as 50 and, for example, in GCL on the ARM architecture the value is indeed only 64.
- **Page 45**: "But its complexity should ~~better~~ be hidden."
- **Page 52**: There's a superfluous word in the C code. Should be:

```
struct node {
  int value;
  struct node *left;
  struct node *right;
}
```

- **Page 80**: "ACROSS works with arbitrary *sequences* (see Chapter 7)" should be "ACROSS works with arbitrary *vectors* (see Chapter 5)." And "strings are sequences," although technically not wrong, should have been "strings are vectors" here (because AREF, of course, doesn't work with arbitrary sequences).
- **Page 82**: The second version of JOIN doesn't handle empty lists properly.[2] If you need that, the easiest way to fix it would be like so:

```
(defun join (separator list)
  (with-output-to-string (out)
    (loop (princ (or (pop list) "") out)                    ;; <-- changed
          (unless list (return))
          (princ separator out))))
```

Or, a bit more chatty but avoiding the OR test inside the loop, like so:

```
(defun join (separator list)
  (with-output-to-string (out)
    (when list
      (princ (pop list) out))
    (loop (unless list (return))
          (princ separator out)
          (princ (pop list) out))))
```

Note that the point of this example in the book was to avoid the extended form of LOOP. If you're not at war with LOOP, there are of course several other alternatives you could employ.

- **Page 85**: Replace the link in footnote 31 with this one: http://xach.com/rpw3/articles/nsKdnbiNlOuNCdLcRVn-pQ%40speakeasy.net.html.
- **Page 93**: The example at the top of the page should look like so:

```
(defconstant +mod+ (expt 2 32))
(defun times-mod (x y) (mod (* x y) +mod+))
(defun times-rem (x y) (rem (* x y) +mod+))
```

If you call these functions with the arguments -58 and 74051161, then TIMES-REM yields -42 while TIMES-MOD gives you 4294967254.

[1] You can, for example, also write a nicer version of DIGITAL-ROOT (page 79) using DIGIT-CHAR-P. For this and similar improvements see the updated code examples at http://weitz.de/cl-recipes/code.zip.

[2] You won't get an error, but the return value will be the string "NIL", which is most likely not what you wanted to see.

- **Page 98**: "An integer, which is always *smaller* than the correct result" should be "an integer that is always *smaller* than the correct result."
- **Page 109**: Mathematically, there's no difference, but in order to be in sync with the footnote, the code at the top of the page should look like this:

```
(1+ (exp (* pi #c(0 1))))
```

- **Page 121**: The initial value (see footnote 8) is not really necessary (although it of course doesn't hurt). `REDUCE` will do the right thing anyway becuase one is the identity element of multiplication and thus `(*)` evaluates to `1`.
- **Page 133**: Replace "suprise" with "surprise."
- **Page 138**: In the REPL interaction, there's one line missing:

```
* (gethash 'superman *h*)
DUCKBURG
T                                                    ;; <- this one
```

- **Page 147**: The second sentence in the first paragraph should be: "You are not allowed to add or remove *entries* while you're in the middle of an iteration through the hash table."
- **Page 151**: You *can* use one of the standard hash tests, namely `EQUALP`. (But let's, for the sake of the example, assume you don't want that, maybe for performance reasons.)
- **Page 155**: $\log_{1.5} \frac{1000000}{16}$ is approximately 27.2 and not 27.4.
- **Page 193**: Rows two and three were swapped in the first column of the table. They should look like so:

`(maplist f list-1 list-2)`	`(loop for x-1 on list-1`
	`  for x-2 on list-2`
	`  collect (funcall f x-1 x-2))`

`(mapcan f list-1 list-2)`	`(loop for x-1 in list-1`
	`  for x-2 in list-2`
	`  nconc (funcall f x-1 x-2))`

- **Page 199**: At the first bullet point, make that "as it *is* obvious."
- **Page 213**: Parts of the paragraphs at the end should be replaced with this text:

  "[...] And names are, of course, symbols,[3] so that, for example, the function defined with `DEFUN` in the preceding example has as its name the symbol `FOO`.

  But how do you get the actual function (as an object), given its name? That's what (the special operator) `FUNCTION` is for; it accepts as its only argument a function name[4] and returns the corresponding function. [...]"
- **Page 218**: It might be worth noting that `WITH-STANDARD-IO-SYNTAX` sets the global special variable `*READ-EVAL*` (see Recipe 8-3) to its default value `T`. This can be a problem if you're reading untrusted data—see the example at the end of page 110.
- **Page 219–221**: Replace `(COPY-READTABLE)` with `(COPY-READTABLE NIL)` (all occurrences).
- **Page 238**: "If your existing stream has a fill pointer" should be "if your existing string has a fill pointer."
- **Page 246**: `"XZYWV"` should be `"XYZWV"`.

[3] But see also page 284 which shows another kind of function names.

[4] It'll also accept *lambda expressions*, but you'll probably never need that. At least not explicitly. Implicitly, this is handled by the `LAMBDA` *macro*, but that's too much detail for this recipe.

- **Page 248**: "For you own classes" should of course be "for your own classes."
- **Page 249**: `(SETQ *PRINT-READABLY* T)` of course returns `T` and not `NIL`.
- **Page 261**: Replace "section 22.2.22" with "section 22.2.2."
- **Page 273**: "Keyword Dames" (first line of page) should obviously be "Keyword Names." (Yikes!)
- **Page 274**: Remove the "(the first time `count` is called)" part.
- **Page 275**: The first paragraph should be replaced with this one: "One subtle difference between C and the COMMON LISP code is that in Lisp the variable is initialized when the function definition is loaded. This might be relevant if the initialization process is complicated (which it can't be in C), but there are, of course, ways to postpone initialization if necessary."
- **Page 288**: The symbol return by the `DEFUN` form (penultimate line on the page) should, of course, be `CDR-ASSOC` and not `MY-CDR-ASSOC`.
- **Page 301**: Replace the semicolon behind "standardized" with a comma.
- **Page 350**: Replace "`fourty-two`" with "`forty-two`."
- **Page 377**: Replace "if we hadn't use" with "if we hadn't used."
- **Page 414**: In the output of the last example on the page, the first three strings should end with double quotes.
- **Page 432**: "Streams in COMMON LISP are instances of CLOS classes." While this is technically not wrong,[5] it might be misleading: one might be tempted to think that streams are instances of *standard classes*.

  However, the result of evaluating

  ```
  (with-input-from-string (s "Frunobulax")
    (class-of (class-of s)))
  ```

  will be the class `STANDARD-CLASS` in *some* implementations, whereas in some others it won't. According to the HyperSpec, the class `STREAM` is a *system class*. See footnote 11 on page 368 of *Common Lisp Recipes* for more on this.

  For a critique of Gray streams, see http://franz.com/support/documentation/current/doc/streams.htm#gray-stream-problems-2.
- **Page 463**: Replace "thatyou" with "that you."
- **Page 464**: In order to avoid misunderstandings, I probably should have written that an arrow means "contains."
- **Page 475**: Replace "youlove" with "you love."
- **Page 477**: The debugger should report the value of `C` as 529 and not as 0.
- **Page 487**: In the footnote, replace "us" with "use."
- **Page 515**: The correct definitions for `TEST-1` and `TEST-2` should look like so:

  ```
  (defun test-1 (max)
    (loop for i below max
          sum (aref *a* i)))

  (defun test-2 (max)
    (declare (optimize (safety 0)))
  ```

[5] It can't really be wrong, because the wording is fuzzy to begin with. The term "CLOS" doesn't even appear anywhere in the standard (except for the historical remarks in section 1.1.2).

```
  (loop for i below max
        sum (aref *a* i)))
```

- **Page 529**: Replace "This is for floats" with "This is correct for floats."
- **Page 548**: In the bottom left corner of the graph we should have 39 and 38 instead of 40 and 39.
- **Pages 586 and 587**: The correct URLs to enter (bottom of page 586 and top of page 587) are these two:

```
http://127.0.0.1:4242/foo
http://127.0.0.1:4242/foobarbaz
```

- **Page 599**: Replace "complete" with "completely."
- **Page 613**: Replace "At first, This might" with "At first, this might."
- **Page 681**: The reference to Recipe 10-10 was meant to be a link to Recipe 22-1.

Thanks to Rainer Joswig, Philipp Marek, and Aaron Chen for many corrections and suggestions and to Chun Tian (binghe), Tim Daly, Jens Teich, Arthur Lemmens, Walter Pelissero, Matthieu Peeters, Douglas Fields, and Pierpaolo Bernardi.

Last update: November 27, 2017.