Functional Programming

Andrew Jones @andrew_jones

Thom Leggett @thomleggett

BrisFunctional

@brisfunctional

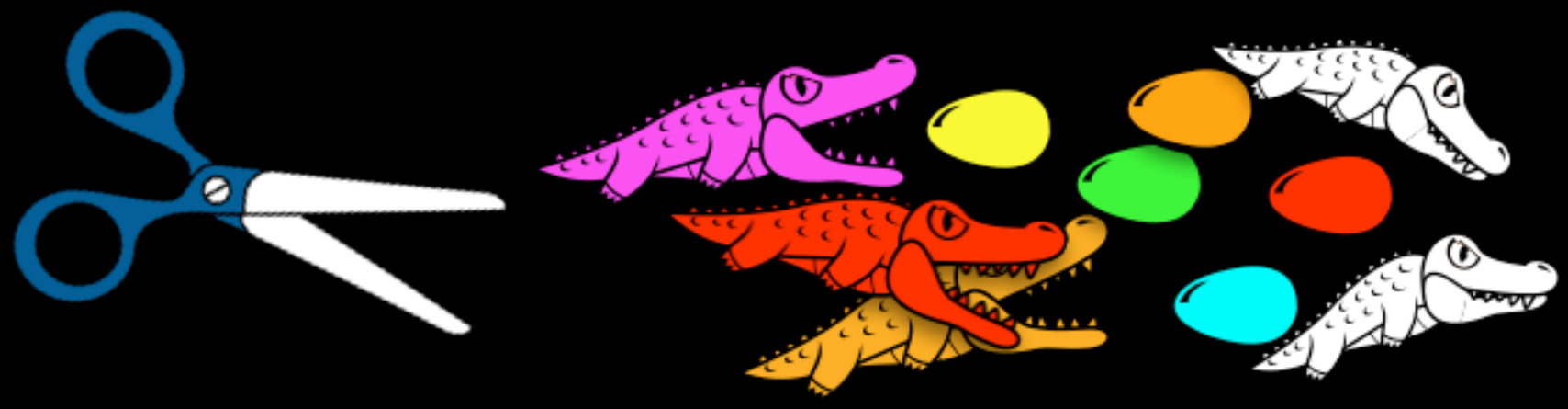
brisfunctional.github.com

Project Euler

JSON parser

Noughts and Crosses AI

http://www.flickr.com/photos/dippy_duck/7210706592/



Alligator Eggs

http://www.flickr.com/photos/14591708@N00/6878586390/

What is Functional Programming?

Functional

Disfunctional

Functional

Disfunctional

Functional

C#

Disfunctional

Functional

C#

Disfunctional

Functional

C#

Haskell

Disfunctional

Functional

C#

Haskell

Javascript

Disfunctional

Functional

C#

Haskell

Javascript

C

Disfunctional

Functional

C#

Haskell

Pascal

Javascript

C

Disfunctional

Java

Functional

C#

Haskell

Pascal

Javascript

C

Disfunctional

Java

Functional

C#

Haskell

Pascal

Javascript

C

Erlang

Disfunctional

Java

Functional

C#

Haskell

Pascal

Javascript

Clojure

C

Erlang

Disfunctional

Java

SQL

Functional

C#

Haskell

Pascal

Javascript

Clojure

C

Erlang

Python

Disfunctional

Java

SQL

Functional

C# → Haskell

Pascal

Javascript

Clojure

C

Erlang

Ruby

Python

Disfunctional

Java

SQL

Functional

C#

Haskell

Pascal

Javascript

Clojure

C

Erlang

Ruby

Python

Disfunctional

Java

SQL

Functional

C#

Haskell

Pascal

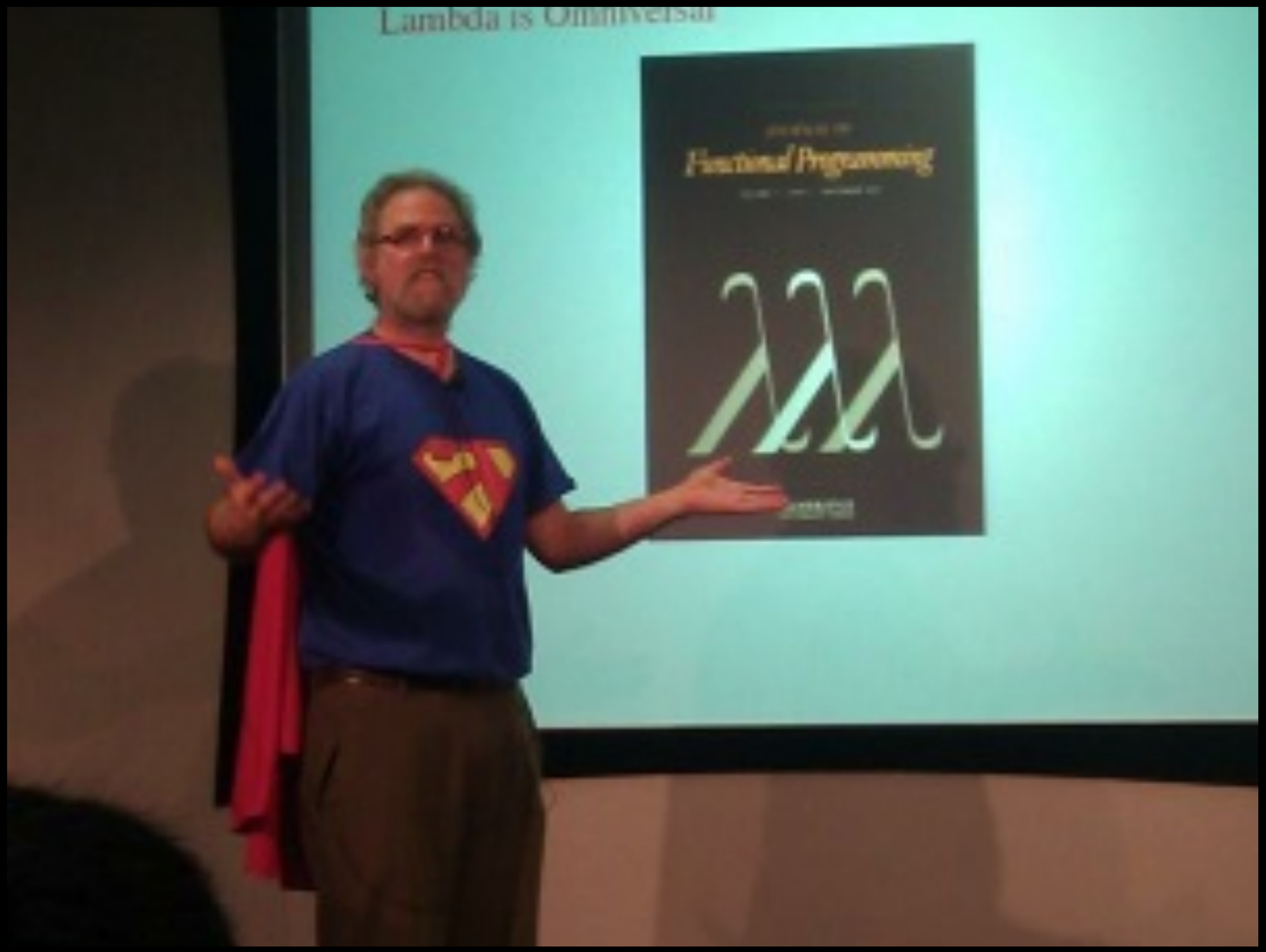
Javascript

Clojure

C

PHP

Erlang



http://alblue.bandlem.com/2012/03/qcon-day-3.html

Immutability


NOKIA
LATEST MIXES
mix radio
Tap to play or press and hold to make available offline.
14:34
Ones to Watch
CANARY WHARF
UK Asian Music Awards 2012
Dubstep
Orbital Recommends


SAMSUNG
8:48 PM
8:48 PM
Tue. May 8
27°C
AccuWeather.com
New Orleans
Cloudy
05/08/2012 7:05 PM
Google
ChatON
Camera
Samsung Apps
Play Store
Phone
Contacts
Messaging
Internet
Apps

```
1 public class Mutable {
2     private List<Sheep> herd;
3
4     public List<Sheep> getHerd() {
5         return herd;
6     }
7 }
```

```
1 public class Immutable {
2     private final List<Sheep> herd;
3
4     public List<Sheep> getHerd() {
5         return Collections.unmodifiableList(herd);
6     }
7
8 }
```

Referential

Transparency

http://www.flickr.com/photos/cruadin/298374640/sizes/l/in/photostream/ http://www.flickr.com/photos/jsutcliffe/5916342650/sizes/l/in/photostream/

http://www.flickr.com/photos/cruadin/298374640/sizes/l/in/photostream/

http://www.flickr.com/photos/jsutcliffe/5916342650/sizes/l/in/photostream/

http://www.flickr.com/photos/cruadin/298374640/sizes/l/in/photostream/

http://www.flickr.com/photos/jsutcliffe/5916342650/sizes/l/in/photostream/

http://www.flickr.com/photos/cradlehall/3583049156/sizes/l/in/photostream/

http://www.flickr.com/photos/cruadin/298374640/sizes/l/in/photostream/

http://www.flickr.com/photos/jsutcliffe/5916342650/sizes/l/in/photostream/

http://www.flickr.com/photos/cradlehall/3583049156/sizes/l/in/photostream/

http://www.flickr.com/photos/cruadin/298374640/sizes/l/in/photostream/

http://www.flickr.com/photos/jsutcliffe/5916342650/sizes/l/in/photostream/

http://www.flickr.com/photos/cradlehall/3583049156/sizes/l/in/photostream/

http://www.flickr.com/photos/cruadin/298374640/sizes/l/in/photostream/

http://www.flickr.com/photos/jsutcliffe/5916342650/sizes/l/in/photostream/

http://www.flickr.com/photos/cradlehall/3583049156/sizes/l/in/photostream/

http://www.flickr.com/photos/cruadin/298374640/sizes/l/in/photostream/

http://www.flickr.com/photos/jsutcliffe/5916342650/sizes/l/in/photostream/

First Class Functions

http://www.flickr.com/photos/22748341@N00/2789592060/sizes/l/in/photostream/

```
1 function shearHerd (herd) {
2     for (var sheep in herd) {
3         shear(sheep);
4     }
5 }
6
7 shearHerd(sheepHerd);
```

http://www.flickr.com/photos/amypalko/3607158016/

http://www.flickr.com/photos/emuphoto/206196733

```
1 function map (herd, action) {
2     for (var animal in herd) {
3         action(animal);
4     }
5 }
6
7 map(sheepHerd, shear);
8 map(cowHerd, milk);
```

Examples

LZW in C

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

/* -------- aux stuff ---------- */
void* mem_alloc(size_t item_size, size_t n_item)
{
    size_t *x = calloc(1, sizeof(size_t)*2 + n_item * item_size);
    x[0] = item_size;
    x[1] = n_item;
    return x + 2;
}

void* mem_extend(void *m, size_t new_n)
{
    size_t *x = (size_t*)m - 2;
    x = realloc(x, sizeof(size_t) * 2 + *x * new_n);
    if (new_n > x[1])
        memset((char*)(x + 2) + x[0] * x[1], 0, x[0] * (new_n - x[1]));
    x[1] = new_n;
    return x + 2;
}

inline void _clear(void *m)
{
    size_t *x = (size_t*)m - 2;
    memset(m, 0, x[0] * x[1]);
}

#define _new(type, n)   mem_alloc(sizeof(type), n)
#define _del(m)     { free((size_t*)(m) - 2); m = 0; }
#define _len(m)     *((size_t*)m - 1)
#define _setsize(m, n)  m = mem_extend(m, n)
#define _extend(m)  m = mem_extend(m, _len(m) * 2)


/* ----------- LZW stuff -------------- */
typedef uint8_t byte;
typedef uint16_t ushort;

#define M_CLR   256 /* clear table marker */
#define M_EOD   257 /* end-of-data marker */
#define M_NEW   258 /* new code index */

/* encode and decode dictionary structures.
   for encoding, entry at code index is a list of indices that follow current one,
   i.e. if code 97 is 'a', code 387 is 'ab', and code 1022 is 'abc',
   then dict[97].next['b'] = 387, dict[387].next['c'] = 1022, etc. */
typedef struct {
    ushort next[256];
} lzw_enc_t;

/* for decoding, dictionary contains index of whatever prefix index plus trailing
   byte.  i.e. like previous example,
    dict[1022] = { c: 'c', prev: 387 },
    dict[387]  = { c: 'b', prev: 97 },
    dict[97]   = { c: 'a', prev: 0 }
   the "back" element is used for temporarily chaining indices when resolving
   a code to bytes
 */
typedef struct {
    ushort prev, back;
    byte c;
} lzw_dec_t;

byte* lzw_encode(byte *in, int max_bits)
{
    int len = _len(in), bits = 9, next_shift = 512;
    ushort code, c, nc, next_code = M_NEW;
    lzw_enc_t *d = _new(lzw_enc_t, 512);

    if (max_bits > 16) max_bits = 16;
    if (max_bits < 9 ) max_bits = 12;

    byte *out = _new(ushort, 4);
    int out_len = 0, o_bits = 0;
    uint32_t tmp = 0;

    inline void write_bits(ushort x) {
        tmp = (tmp << bits) | x;
        o_bits += bits;
        if (_len(out) <= out_len) _extend(out);
        while (o_bits >= 8) {
            o_bits -= 8;
            out[out_len++] = tmp >> o_bits;
            tmp &= (1 << o_bits) - 1;
        }
    }

    //write_bits(M_CLR);
    for (code = *(in++); --len; ) {
        c = *(in++);
        if ((nc = d[code].next[c]))
            code = nc;
        else {
            write_bits(code);
            nc = d[code].next[c] = next_code++;
            code = c;
        }

        /* next new code would be too long for current table */
        if (next_code == next_shift) {
            /* either reset table back to 9 bits */
            if (++bits > max_bits) {
                /* table clear marker must occur before bit reset */
                write_bits(M_CLR);

                bits = 9;
                next_shift = 512;
                next_code = M_NEW;
                _clear(d);
            } else  /* or extend table */
                _setsize(d, next_shift *= 2);
        }
    }

    write_bits(code);
    write_bits(M_EOD);
    if (tmp) write_bits(tmp);

    _del(d);

    _setsize(out, out_len);
    return out;
}
```

LZW in Clojure

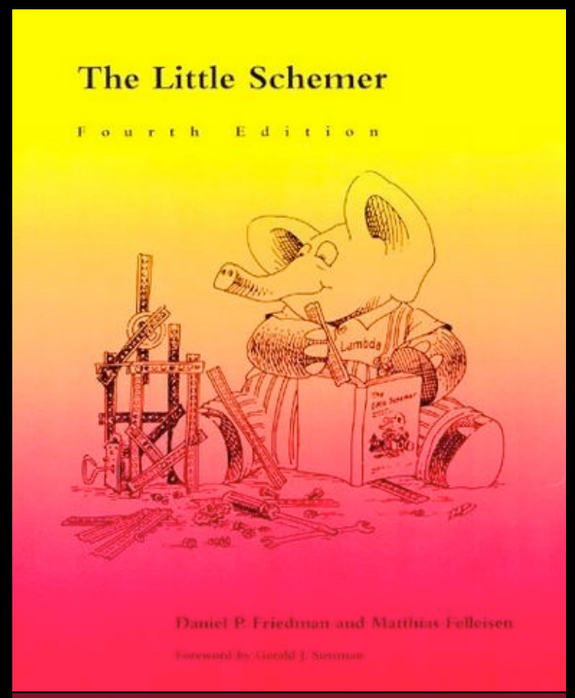
```clojure
(defn make-dict []
  (let [vals (range 0 256)]
    (zipmap (map (comp #'list #'char) vals) vals)))

(defn compress [#^String text]
  (loop [t (seq text)
         r '()
         w '()
         dict (make-dict)
         s 256]
    (let [c (first t)]
      (if c
        (let [wc (cons c w)]
          (if (get dict wc)
            (recur (rest t) r wc dict s)
            (recur (rest t) (cons (get dict w) r) (list c)
              (assoc dict wc s) (inc s))))
        (reverse (if w (cons (get dict w) r) r))))))

(compress "TOBEORNOTTOBEORTOBEORNOT")
```

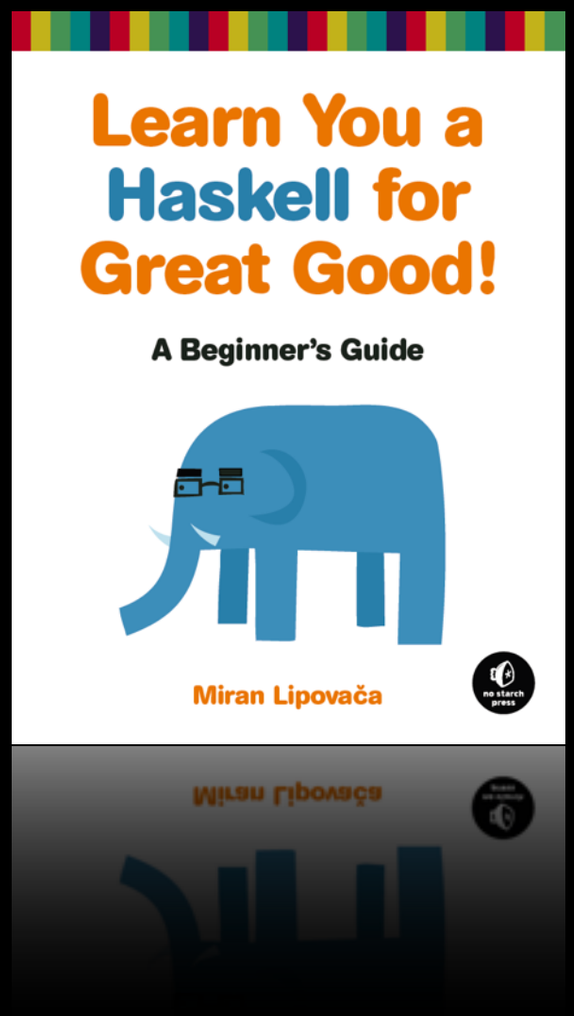
LZW in Haskell

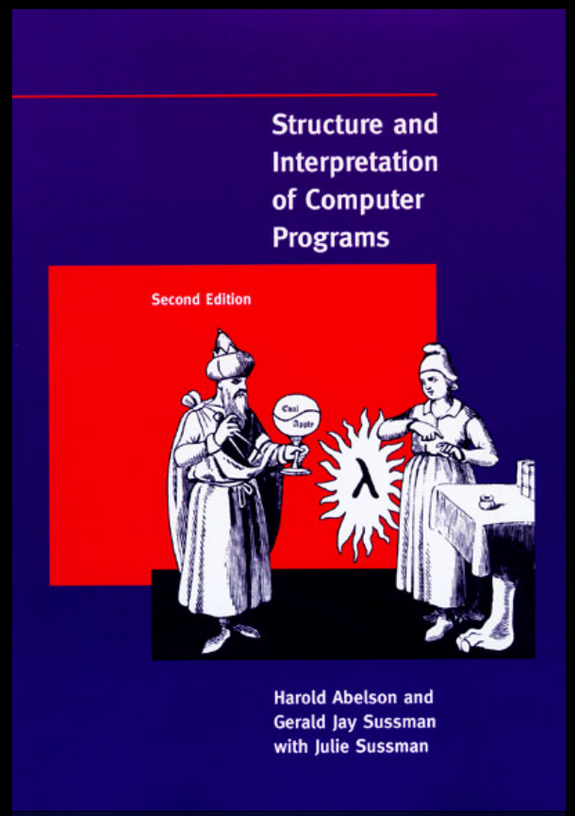
```haskell
import Data.List
import Data.Char
import Data.Maybe

doLZW _ [] = []
doLZW as (x:xs) =  lzw (map return as) [x] xs
   where lzw a w [] = [fromJust $ elemIndex w a]
         lzw a w (x:xs)  | w' `elem` a = lzw a w' xs
                         | otherwise   = fromJust (elemIndex w a) :
                                                 lzw (a++[w']) [x] xs
               where w' = w++[x]
```

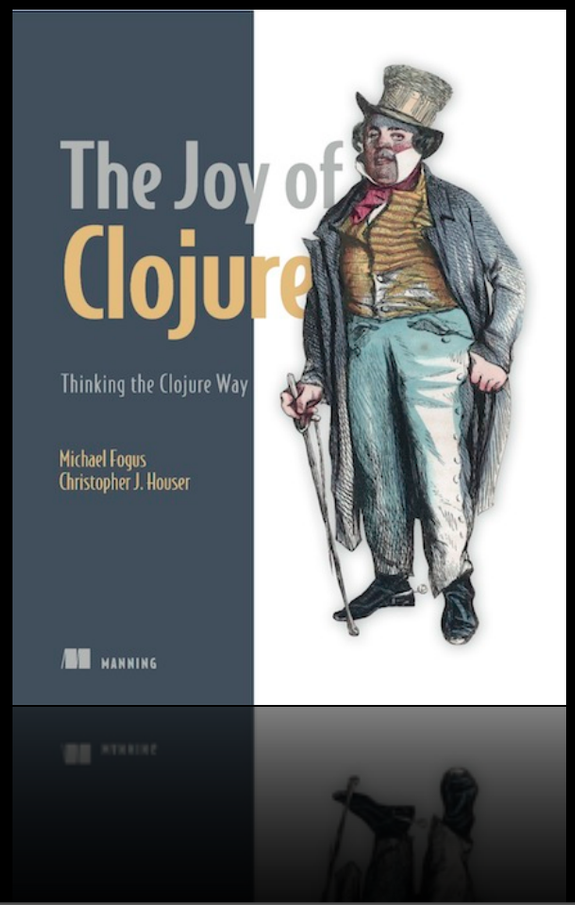

ERLANG

Recommended Reading


The Little Schemer
Fourth Edition
Daniel P. Friedman and Matthias Felleisen


Learn You a Haskell for Great Good!
A Beginner's Guide
Miran Lipovača
no starch press


Structure and Interpretation of Computer Programs
Second Edition
Harold Abelson and
Gerald Jay Sussman
with Julie Sussman


The Joy of Clojure
Thinking the Clojure Way
Michael Fogus
Christopher J. Houser
MANNING

Closing Thought

One thing that makes a programming language 'more powerful' in my opinion is the provision of more ways to factor programs. Or if you prefer, more axes of composition. The more different ways you can compose programs out of subprograms, the more powerful a language is.

Reg Braithwaite - Functional Programming Matters

Thanks!

Andrew Jones @andrew_jones
Thom Leggett @thomleggett