

# Functional Programming

**Andrew Jones** @andrew\_jones  
**Thom Leggett** @thomleggett

# BrisFunctional

@brisfunctional  
brisfunctional.github.com

# Project Euler

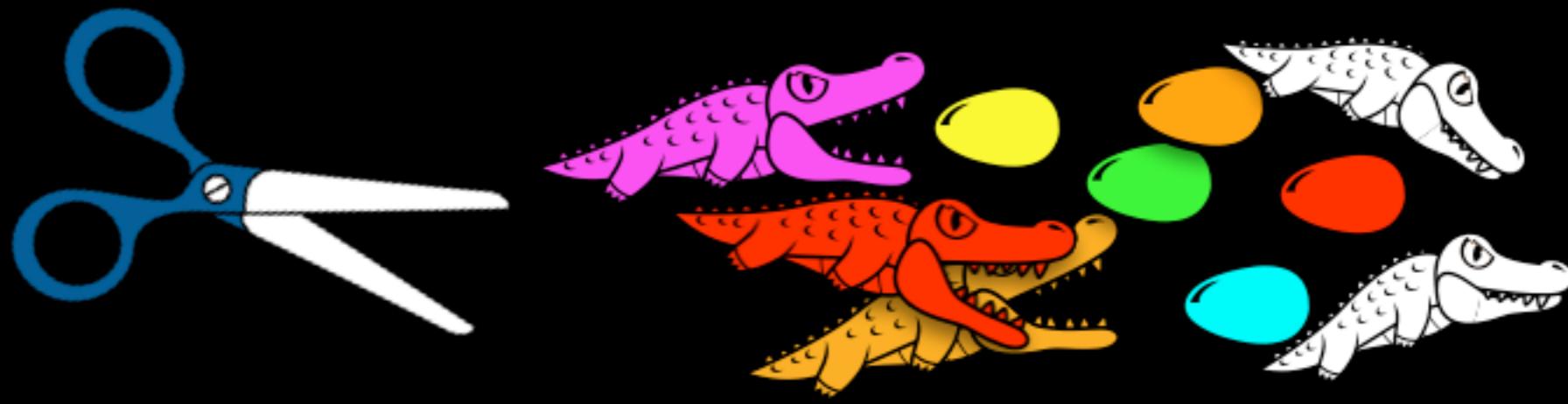
# JSON parser

# Noughts and Crosses

## AI



[http://www.flickr.com/photos/dippy\\_duck/7210706592/](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



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

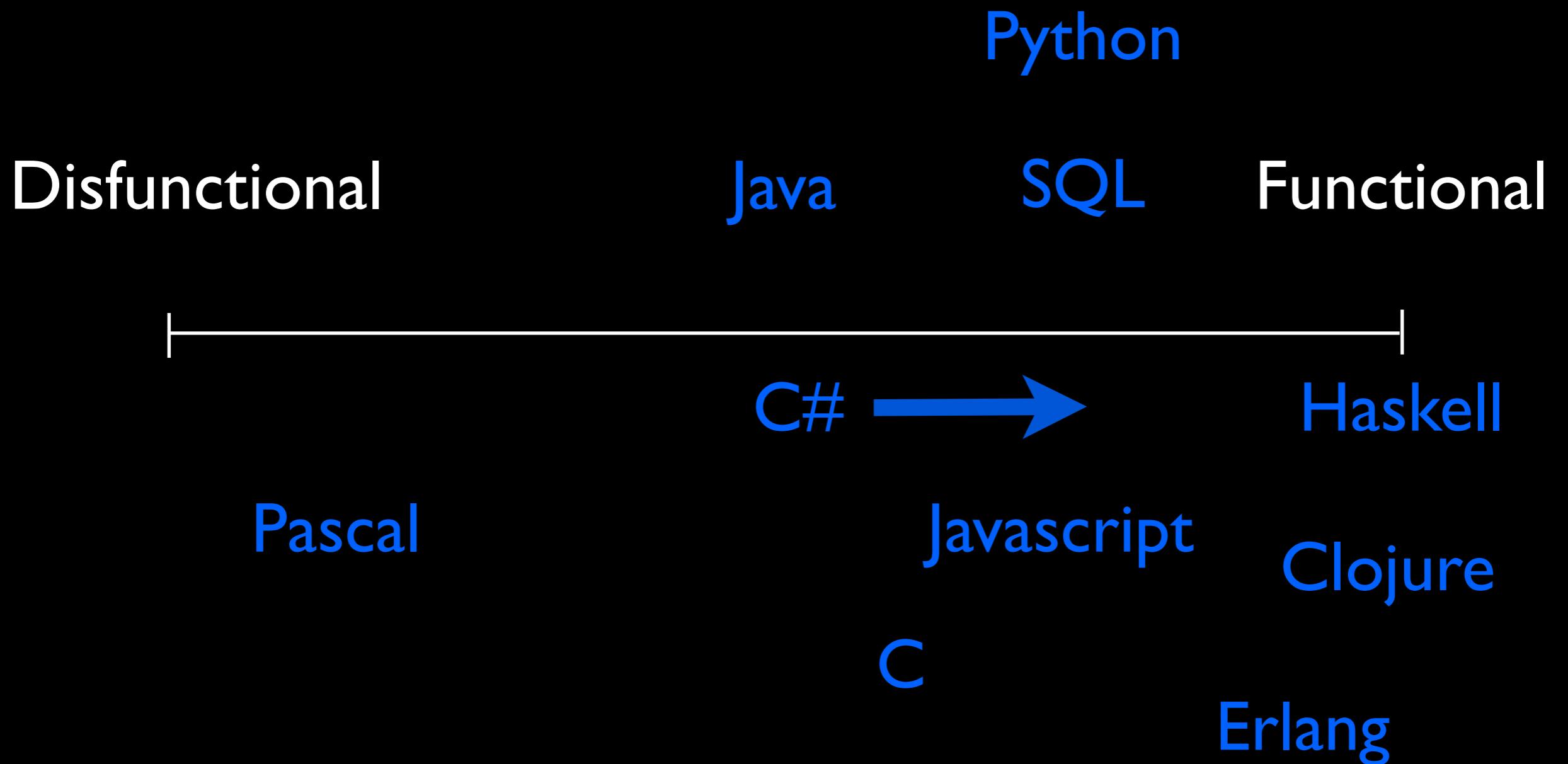
Pascal

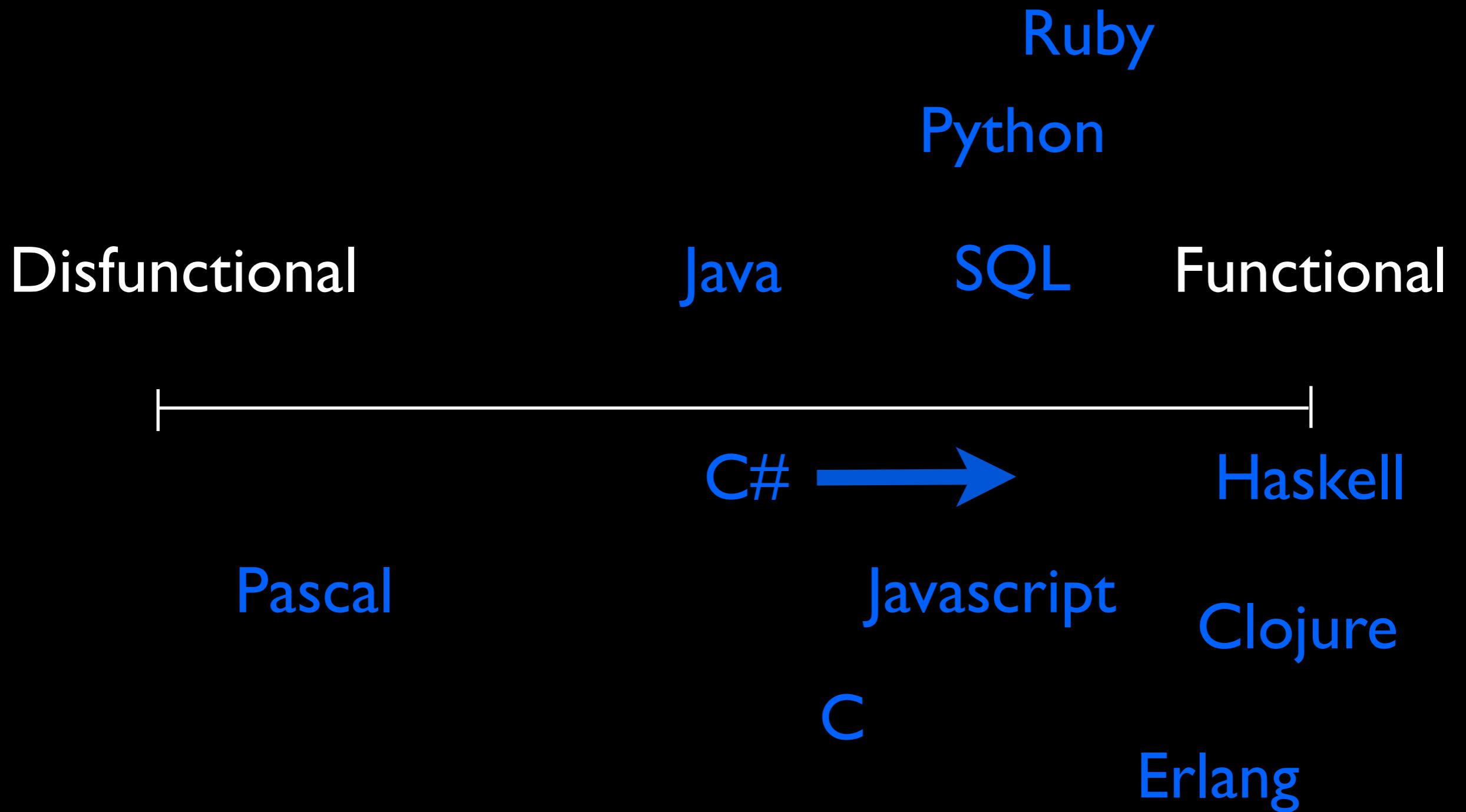
Javascript

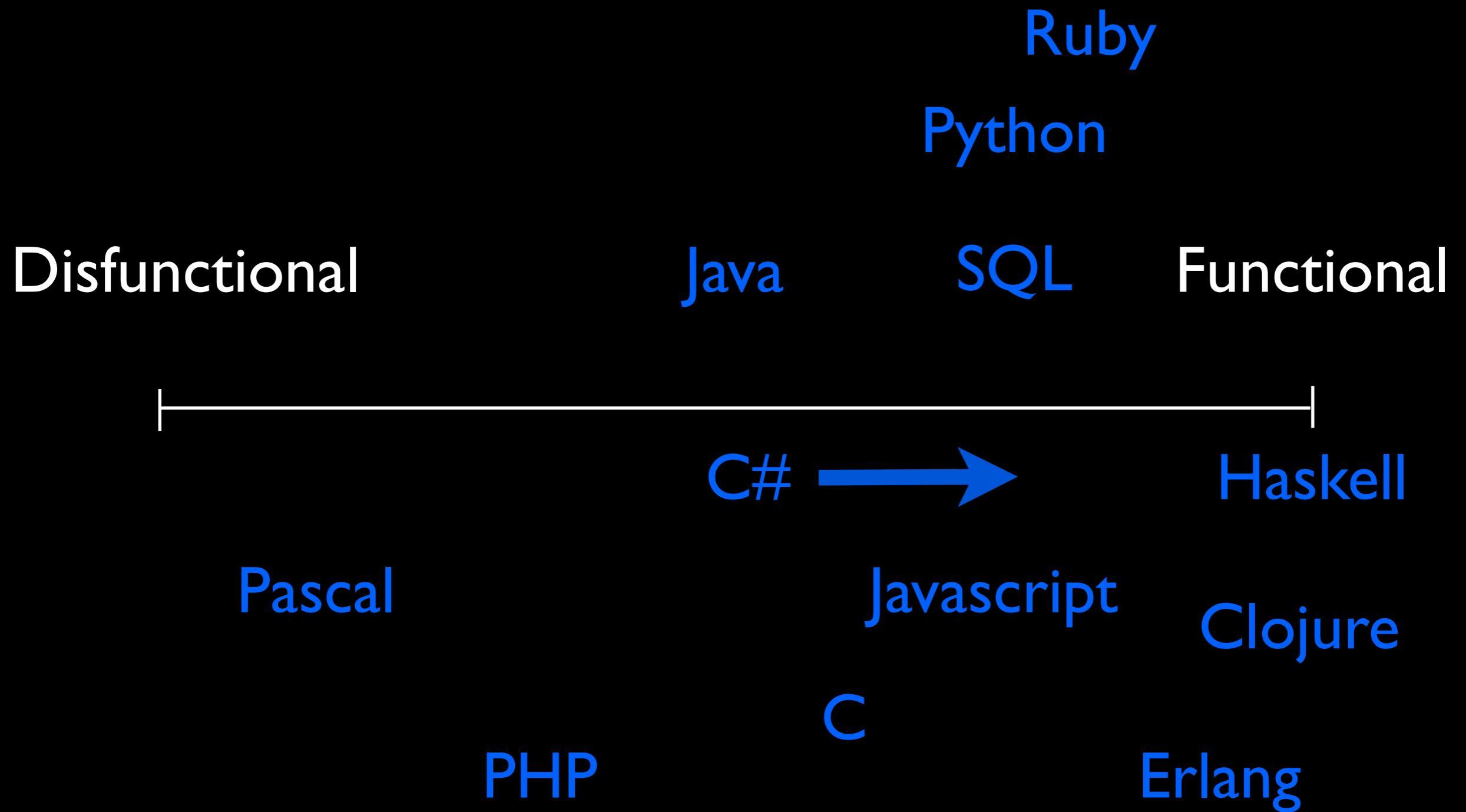
Clojure

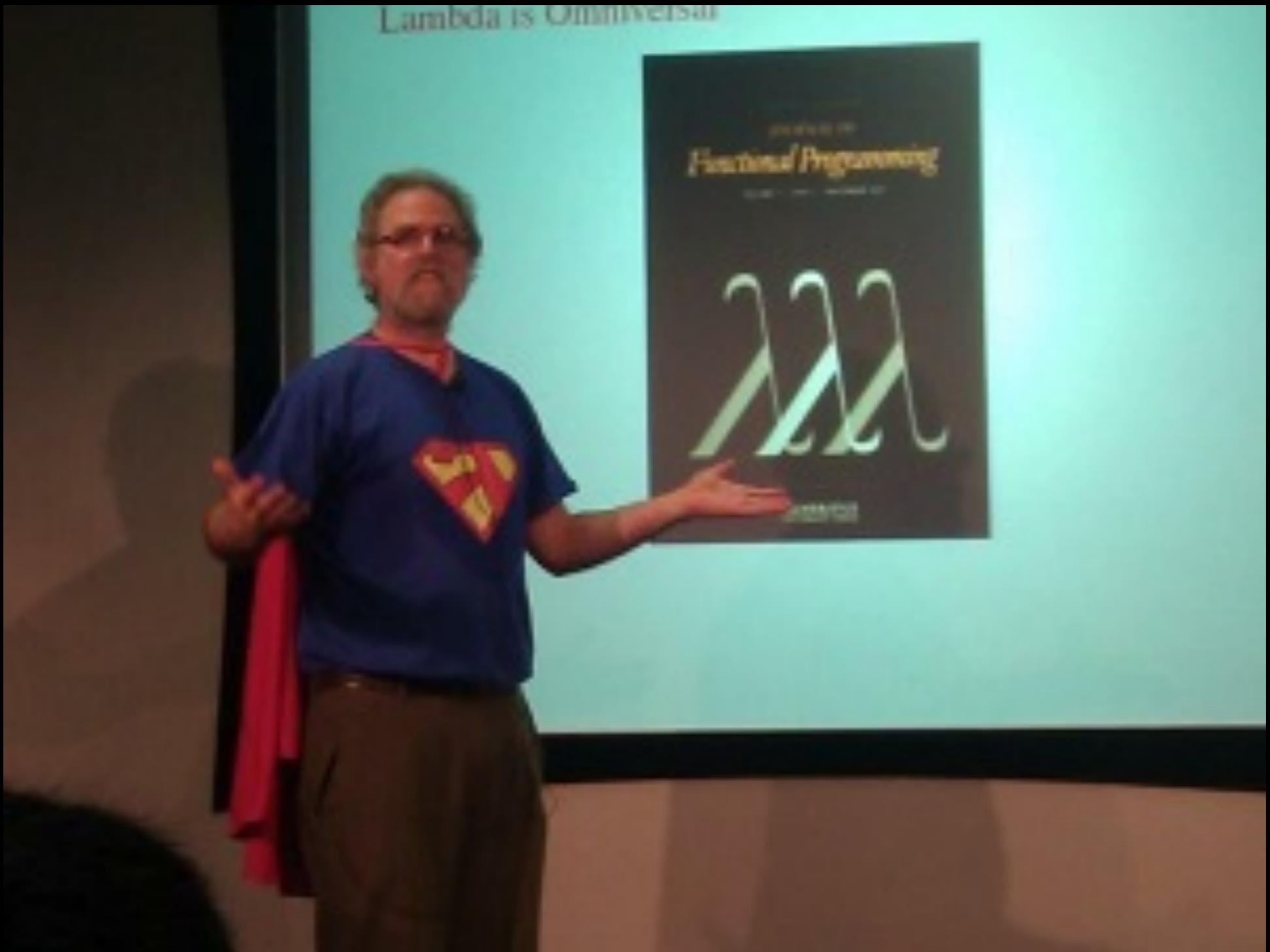
C

Erlang









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

# Immutability





```
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/>

By Cstalla (Own work) [GFDL (<http://www.gnu.org/copyleft/fdl.html>), CC-BY-SA-3.0 (<http://creativecommons.org/licenses/by-sa/3.0/>) or FAL],  
via Wikimedia Commons



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

By Cstalla (Own work) [GFDL (<http://www.gnu.org/copyleft/fdl.html>), CC-BY-SA-3.0 (<http://creativecommons.org/licenses/by-sa/3.0/>) or FAL],  
via Wikimedia Commons

<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/>

By Cstalla (Own work) [GFDL (<http://www.gnu.org/copyleft/fdl.html>), CC-BY-SA-3.0 (<http://creativecommons.org/licenses/by-sa/3.0/>) or FAL],  
via Wikimedia Commons



<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/>

By Cstalla (Own work) [GFDL (<http://www.gnu.org/copyleft/fdl.html>), CC-BY-SA-3.0 (<http://creativecommons.org/licenses/by-sa/3.0/>) or FAL],  
via Wikimedia Commons



<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/>

By Cstalla (Own work) [GFDL (<http://www.gnu.org/copyleft/fdl.html>), CC-BY-SA-3.0 (<http://creativecommons.org/licenses/by-sa/3.0/>) or FAL],  
via Wikimedia Commons



<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/>

By Cstalla (Own work) [GFDL (<http://www.gnu.org/copyleft/fdl.html>), CC-BY-SA-3.0 (<http://creativecommons.org/licenses/by-sa/3.0/>) or FAL],  
via Wikimedia Commons



<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/>

By Cstalla (Own work) [GFDL (<http://www.gnu.org/copyleft/fdl.html>), CC-BY-SA-3.0 (<http://creativecommons.org/licenses/by-sa/3.0/>) or FAL],  
via Wikimedia Commons

# 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

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <stdint.h>
5 #include <unistd.h>
6 #include <fcntl.h>
7 #include <sys/types.h>
8 #include <sys/stat.h>
9
10 /* ----- aux stuff ----- */
11 void* mem_alloc(size_t item_size, size_t n_item)
12 {
13     size_t *x = calloc(1, sizeof(size_t)*2 + n_item * item_size);
14     x[0] = item_size;
15     x[1] = n_item;
16     return x + 2;
17 }
18
19 void* mem_extend(void *m, size_t new_n)
20 {
21     size_t *x = (size_t*)m - 2;
22     x = realloc(x, sizeof(size_t) * 2 + *x * new_n);
23     if (new_n > x[1])
24         memset((char*)(x + 2) + x[0] * x[1], 0, x[0] * (new_n - x[1]));
25     x[1] = new_n;
26     return x + 2;
27 }
28
29 inline void _clear(void *m)
30 {
31     size_t *x = (size_t*)m - 2;
32     memset(m, 0, x[0] * x[1]);
33 }
34
35 #define _new(type, n)    mem_alloc(sizeof(type), n)
36 #define _del(m)          { free((size_t*)(m) - 2); m = 0; }
37 #define _len(m)           *(size_t*)m - 1
38 #define _setsize(m, n)   m = mem_extend(m, n)
39 #define _extend(m)        m = mem_extend(m, _len(m) * 2)
40
41 /* ----- LZW stuff ----- */
42 typedef uint8_t byte;
43 typedef uint16_t ushort;
44
45 #define M_CLR 256 /* clear table marker */
46 #define M_EOD 257 /* end-of-data marker */
47 #define M_NEW 258 /* new code index */
48
49 /* encode and decode dictionary structures.
50  for encoding, entry at code index is a list of indices that follow current one,
51  i.e. if code 97 is 'a', code 387 is 'ab', and code 1022 is 'abc',
52  then dict[97].next['b'] = 387, dict[387].next['c'] = 1022, etc. */
53
54 typedef struct {
55     ushort next[256];
56 } lzw_enc_t;
57
58 /* for decoding, dictionary contains index of whatever prefix index plus trailing
59  byte. i.e. like previous example,
60  dict[1022] = { c: 'c', prev: 387 },
61  dict[387] = { c: 'b', prev: 97 },
62  dict[97] = { c: 'a', prev: 0 }
63  the "back" element is used for temporarily chaining indices when resolving
64  a code to bytes
65 */
66 typedef struct {
67     ushort prev, back;
68     byte c;
69 } lzw_dec_t;
70
71 byte* lzw_encode(byte *in, int max_bits)
72 {
73     int len = _len(in), bits = 9, next_shift = 512;
74     ushort code, c, nc, next_code = M_NEW;
75     lzw_enc_t *d = _new(lzw_enc_t, 512);
76
77     if (max_bits > 16) max_bits = 16;
78     if (max_bits < 9) max_bits = 12;
79
80     byte *out = _new(ushort, 4);
81     int out_len = 0, o_bits = 0;
82     uint32_t tmp = 0;
83
84     inline void write_bits(ushort x) {
85         tmp = (tmp << bits) | x;
86         o_bits += bits;
87         if (_len(out) <= out_len) _extend(out);
88         while (o_bits >= 8) {
89             o_bits -= 8;
90             out[out_len++] = tmp >> o_bits;
91             tmp &= (1 << o_bits) - 1;
92         }
93     }
94
95     //write_bits(M_CLR);
96     for (code = *(in++); --len; ) {
97         c = *(in++);
98         if ((nc = d[code].next[c]))
99             code = nc;
100        else {
101            write_bits(code);
102            nc = d[code].next[c] = next_code++;
103            code = c;
104        }
105
106        /* next new code would be too long for current table */
107        if (next_code == next_shift) {
108            /* either reset table back to 9 bits */
109            if (++bits > max_bits) {
110                /* table clear marker must occur before bit reset */
111                write_bits(M_CLR);
112
113                bits = 9;
114                next_shift = 512;
115                next_code = M_NEW;
116                _clear(d);
117            } else /* or extend table */
118                _setsize(d, next_shift *= 2);
119        }
120    }
121
122    write_bits(code);
123    write_bits(M_EOD);
124    if (tmp) write_bits(tmp);
125
126    _del(d);
127
128    _setsize(out, out_len);
129    return out;
130 }

```

# LZW in Clojure

```

1 (defn make-dict []
2   (let [vals (range 0 256)]
3     (zipmap (map (comp #'list #'char) vals) vals)))
4
5 (defn compress [#^String text]
6   (loop [t (seq text)
7         r '()
8         w '()
9         dict (make-dict)
10        s 256]
11     (let [c (first t)]
12       (if c
13           (let [wc (cons c w)]
14             (if (get dict wc)
15                 (recur (rest t) r wc dict s)
16                 (recur (rest t) (cons (get dict w) r) (list c)
17                         (assoc dict wc s) (inc s))))
18           (reverse (if w (cons (get dict w) r) r))))))
19
20 (compress "TOBEORNOTTOBEORTOBEORNOT")

```

# LZW in Haskell

```
1 import Data.List
2 import Data.Char
3 import Data.Maybe
4
5 doLZW _ [] = []
6 doLZW as (x:xs) = lzw (map return as) [x] xs
7   where lzw a w [] = [fromJust $ elemIndex w a]
8         lzw a w (x:xs) | w' `elem` a = lzw a w' xs
9                     | otherwise    = fromJust (elemIndex w a) :
10                           lzw (a++[w']) [x] xs
11   where w' = w++[x]
```

# Recommended Reading

## The Little Schemer

Fourth Edition



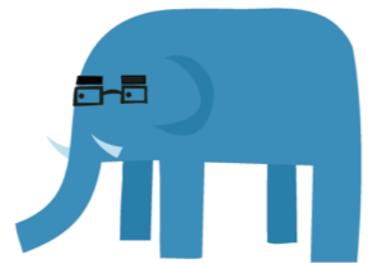
Daniel P. Friedman and Matthias Felleisen

Foreword by Gerald J. Sussman

Copyright © 2008 Daniel P. Friedman and Matthias Felleisen  
All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in whole or in part, without the prior written permission of the author, excepting brief quotations in critical articles or reviews.

## Learn You a Haskell for Great Good!

A Beginner's Guide



Miran Lipovača



Miran Lipovača



Structure and Interpretation of Computer Programs

Second Edition



Harold Abelson and  
Gerald Jay Sussman  
with Julie Sussman

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



# 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