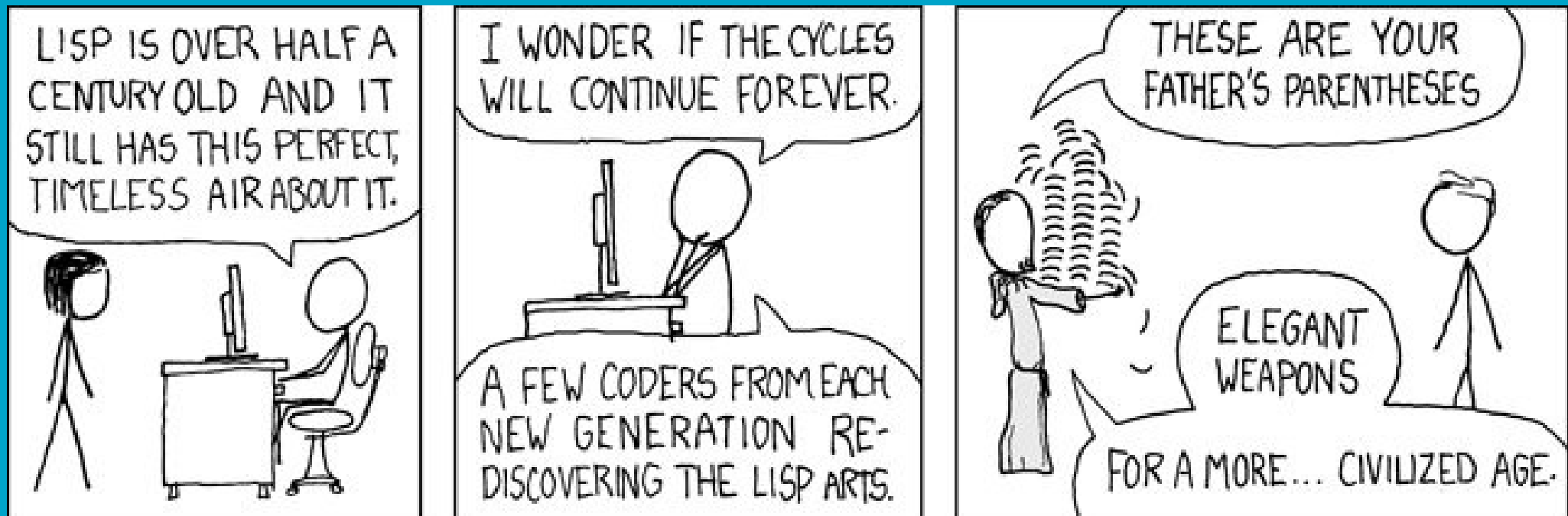


Scheme  
Or:  
How I Learned to Stop Worrying  
and Love Parentheses.

Dan Cosser  
10 March '08



# Comedy



# What is Scheme?

Scheme is an  
implementation of LISP

# What is LISP?

Lisp is...

- ★ A multi paradigm language
  - ★ Functional
  - ★ Procedural
  - ★ Reflective
  - ★ Also has OO features
- ★ Based around linked lists
- ★ Nothing like what you're probably used to

# Some history

- ★ An old, mature language
- ★ Inspired Alan Kay, along with many other language designers
- ★ Broke programming theory into its base parts

# An aside: Polish Notation

- ★ Also known as 'Prefix Notation'
- ★ Instead of:  $1 + 2$
- ★ You have:  $+ 1 2$
- ★ Image '+' as a method; you wish to apply the function 'add' to the arguments '1' and '2'
- ★ Prevents any kind of ambiguousness
- ★ No operator precedence

# An aside: S-Expressions

- ★ Everything in lisp is arranged in S-Expressions
- ★ 'Atoms' arranged in 'Lists'
- ★ The atoms '1', '2', and '3' in a list: `( 1 2 3 )`
- ★ Atoms are strings, numbers, symbols, or anything, really
- ★ Nestification: `( 1 ( " c h e e s e " x y z ' E ) # \ g )`

# Some Scheme

The background features a bright yellow sky at the top. Below it are several layers of wavy, rounded shapes. The top layer is a light blue, followed by a layer of brown, then a layer of medium blue, and finally a solid dark blue at the bottom. The waves in the blue layers are out of phase with each other, creating a layered, water-like effect.

Functions are just Atoms

# Some Scheme

To call a function:

just put the function as the first atom in a list.

The remaining atoms are the arguments.

# Dotted Pairs

★ (define x (cons 1 2)) => (1 . 2)

★ (car x) => 1

★ (cdr x) => 2

★ (define y (cons 1 (cons 2 3)))  
=> (1 . (2 . 3))

★ (car y) => 1

★ (cdr y) => (2 . 3)

★ (car (cdr y)) = 2

# Lists

`(1 2 3 4 5)`

is the same as

`(1 . (2 . (3 . (4 . (5 . ())))))`

You can use `(list)` to quickly create these

# Doing things in order

```
(begin (display "Hello ")  
       (display #\newline)  
       (display "World!"))
```

# Boring Stuff

★ `(if (expression) result-if-true  
          result-if-false)`

★ `(cond ((expression) result-if-true)  
          ((expression) result-if-true)  
          (else result-otherwise))`

# Defining Functions

## Lambda Functions

```
(lambda (x)
  (+ x 1))
```

# Defining Functions

```
(define ++ (lambda (x)
             (+ x 1)))
```

# Let

`let`, `let*`, `let-rec`, `fluid-let`

All ways of making sure your variables  
are in the correct scope

# Recursion

The background of the slide features a stylized landscape. The top portion is a solid yellow sky. Below the sky are several layers of wavy, rounded shapes representing hills and water. The hills are colored in a gradient from light blue to a darker blue, with some brownish-tan shapes interspersed. The bottom portion of the slide is a solid, bright blue area representing water.

Scheme has no loops

# Recursion



There is no cost to use 'tail call recursion' in Scheme.

Except the tax on your brain.

# Recursion

```
(define countdown  
  (lambda (x)  
    (if (= x 0) (display "Boom!")  
        (countdown (- x 1)))))
```

# Recursion

```
(define list-position
  (lambda (o l)
    (let loop ((i 0) (l l))
      (if (null? l) #f
          (if (eqv? (car l) o) i
              (loop (+ i 1) (cdr l)))))))
```

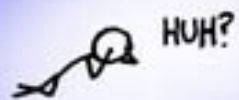
# That Made No Sense

Why should you learn Scheme?

(Or Lisp, or Dylan, or Haskell, or Scala...)

# Comedy

LAST NIGHT I DRIFTED OFF WHILE READING A LISP BOOK.



SUDDENLY, I WAS BATHED IN A SUFFUSION OF BLUE.

AT ONCE, JUST LIKE THEY SAID, I FELT A GREAT ENLIGHTENMENT. I SAW THE NAKED STRUCTURE OF LISP CODE UNFOLD BEFORE ME.



THE PATTERNS AND METAPATTERNS DANCED. SYNTAX FADED, AND I SWAM IN THE PURITY OF QUANTIFIED CONCEPTION. OF IDEAS MANIFEST.

TRULY, THIS WAS THE LANGUAGE FROM WHICH THE GODS WROUGHT THE UNIVERSE.



NO, IT'S NOT.



I MEAN, OSTENSIBLY, YES. HONESTLY, WE HACKED MOST OF IT TOGETHER WITH PERL.