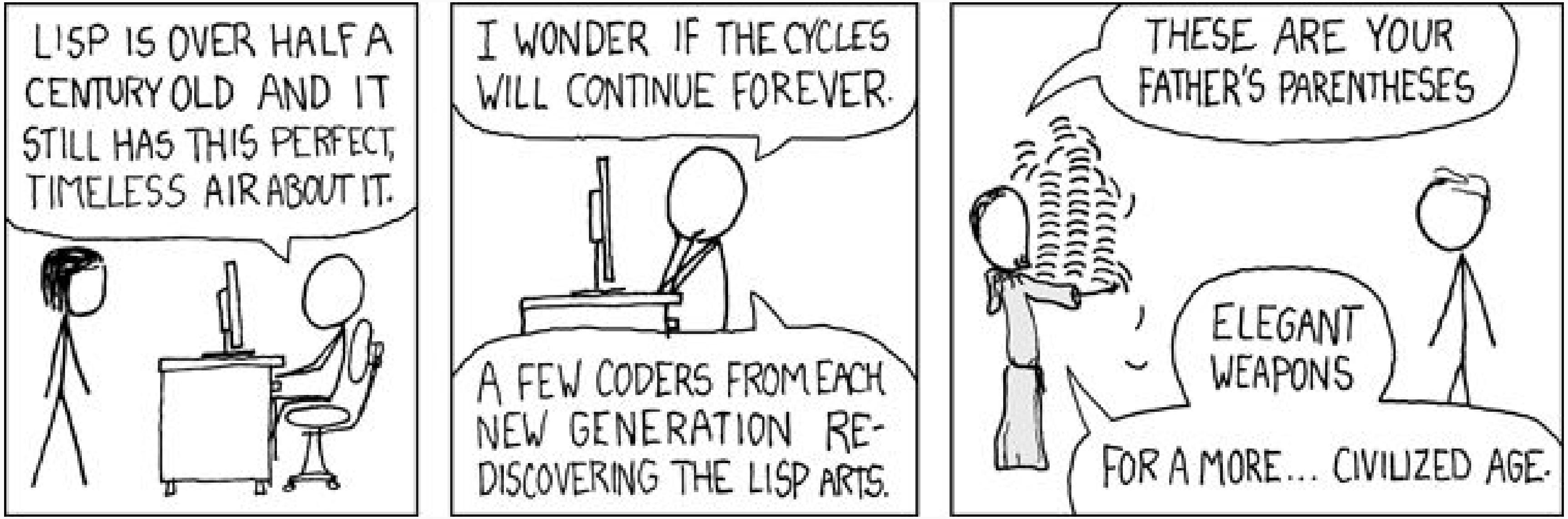
Language Design: The Hard Way

jaywunder/stutter.js


LISP IS OVER HALF A CENTURY OLD AND IT STILL HAS THIS PERFECT, TIMELESS AIR ABOUT IT.
I WONDER IF THE CYCLES WILL CONTINUE FOREVER.
A FEW CODERS FROM EACH NEW GENERATION RE-DISCOVERING THE LISP ARTS.
THESE ARE YOUR FATHER'S PARENTHESES
ELEGANT WEAPONS
FOR A MORE... CIVILIZED AGE.

xkcd.com/297

```
(defun (factorial n)
  (if (== 0 n)
    (return 1)
   else
    (return (* n (factorial (- n 1))))
  )
)

(print (factorial 10))
> 3628800
```

Attempt #1: Rust

?

Attempt #2: Javascript

Javascript

The Details

Step 1: Parsing

1. Read from file
2. Split input string into special characters and tokens
3. Split tokens into a tree of Arrays

Read from file

```
fs.readFileSync(filename, 'UTF-8');
```

Lexer

```
(print 'hello world')
```

```
["(", "print", "'", "hello", "world", "'", ")"]
```

Expression-ize pt.1

- Trees are nested arrays
- Atoms are strings

but…

- Array literals are also arrays
- String literals are also strings

Expression-ize pt.1

Solution:

- ‘#’ syntax
- `Symbol('literal')`

Expression-ize pt.1

```
(print 'hello world')
```

```
["(", "print", Symbol("literal"), "'", "hello", "'", "world", ")"]
```

Expression-ize pt.2

Use classes!

- class Expression
- class Atom
- class Literal

Expression-ize pt.2

```
(print 'hello world')
```

```
[ Atom {name:'print'}, Literal {value: 'hello world'} ]
```

Expressionize Code Recursively

```
if (token === '(') { // start of a new expression

  let newExpression;
  [newExpression, i] = expressionize(stream, i + 1); // use fancy new
destructuring
  expression.push( Expression.from(newExpression) );

} else if (token === ')') { // end of an expression

  if ( returnIndex ) // this needs to work recursively and non-recursively
    return [ expression, i ];
  else
    return expression;
}
```

Expressionize Code Iteratively

```
if (token === '[') { // arrays

  let array = [];
  while(stream[++i] !== ']') { // walk token stream
    array.push(stream[i]); // add tokens to the array
  }

  expression.push( new Literal(arr) );
}
```

Step 2: Evaling

Each token can be one of three cases

- Expression
- Atom
- Literal

Eval Code

```
if ( isExpression(token) )
  // do some magic

if ( isAtom(token) )
  return this.get(token);

if ( isLiteral(token) )
  return token.value;
```

Demo

```
(defun demo (audience)
  (amaze audience))

( demo )
```

Lessons Learned?

Thank you!

jaywunder/stutter.js