RubyLisp

An embeddable Scheme for Ruby and RubyMotion

Dave Astels

But first....

What is Functional Programming?

“programs are executed by evaluating *expressions*"

“typically avoids using mutable state"

“functions are *first-class*"

“automatic memory management"

What is Lisp?

“Lisp is the greatest single programming language ever designed"

– Alan Kay

"Lisp is a programmable programming language."

– John Foderaro

"Lisp isn't a language, it's a building material."

– Alan Kay

"Lisp is worth learning for the profound enlightenment experience you will have when you finally get it; that experience will make you a better programmer for the rest of your days, even if you never actually use Lisp itself a lot."

– Eric Raymond, "How to Become a Hacker"

An Aside

Structure and Interpretation of Computer Programs

Second Edition

Harold Abelson and
Gerald Jay Sussman
with Julie Sussman

SICP

Wednesday, Feb 18, 2015 from 6:00 PM to 9:00 PM (CDT)
and each 3rd Wednesday thereafter for ~10 months.

Brad's Deals
640 N LaSalle St
Suite 460
Chicago, IL 60654

Search for “EventBrite SICP Chicago w/ Dave Astels”

Learn Lisp, something about computation &
data abstraction, and have your mind blown

repeatedly

And, we're back

"Lisp is a programmer amplifier."

– Martin Rodgers

- Originally created in 1958
- LISP = LISt Processing… everything is a value or a list of values and/or lists
- Very popular Artificial Intelligence language
- One of the most influential languages
- Lots of dialects: MacLisp, FranzLisp, CommonLisp, Scheme, Clojure


1954
1960
1965
1970
1975
1980
1985
1990
1995
2000
2001
2002
2003
2004

```
(define (double x)
    (+ x x))

(double 5) ⇒ 10
```

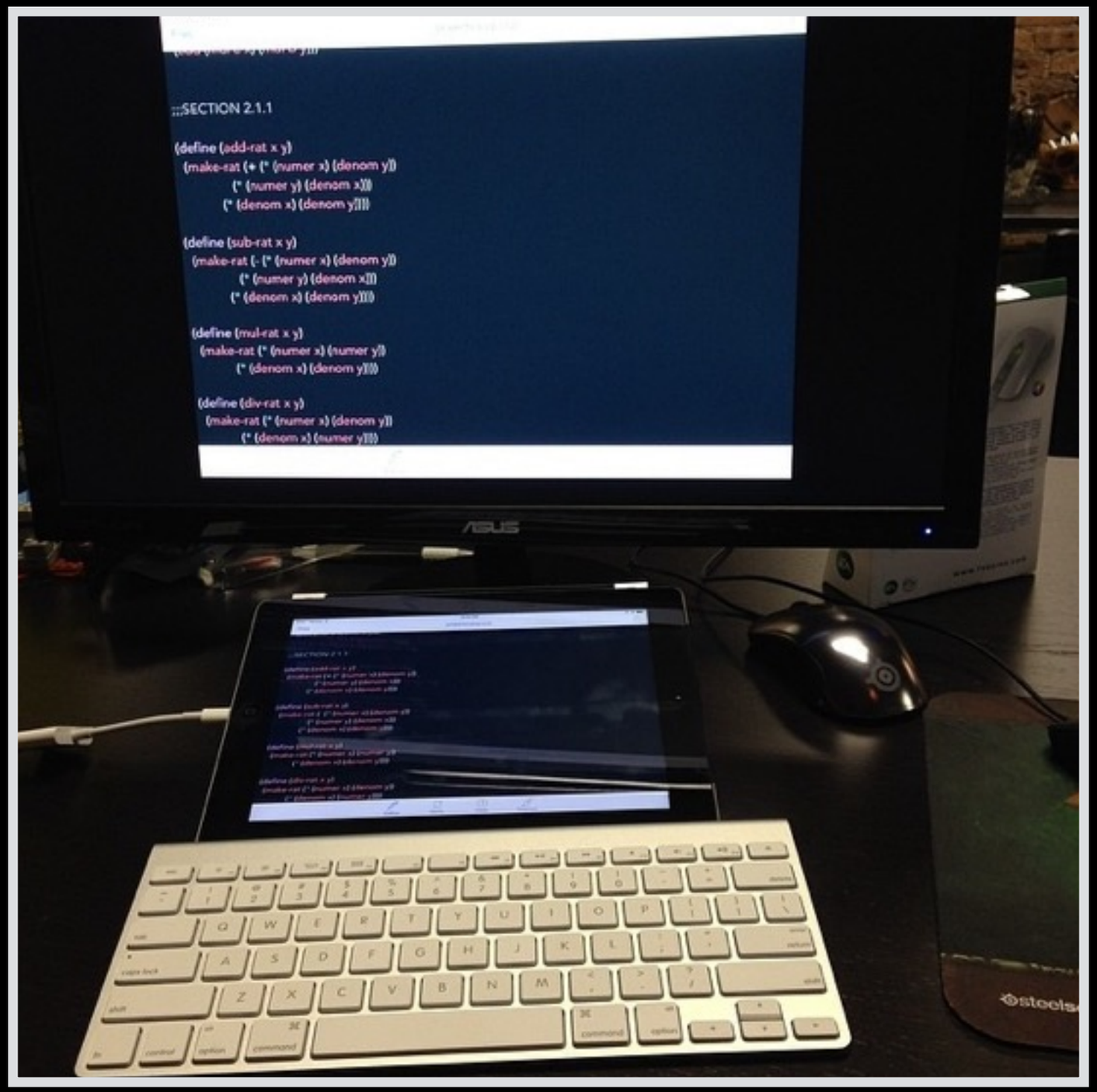
```
(define (fib n)
  (cond ((eq? n 0) 0)
        ((<= n 2) 1)
        (else (+ (fib (- n 1))
                 (fib (- n 2))))))
```

What is RubyLisp?

RubyLisp

- Inspired by and based on MIT Scheme
- 100% implemented in pure Ruby
- A single gem
- Well covered by tests (Ruby and Lisp)
- Easily extendable in Ruby
- Easily integrated into a Ruby app

Motivation


;;;SECTION 2.1.1
(define (add-rat x y)
(make-rat (+ (* (numer x) (denom y))
(* (numer y) (denom x)))
(* (denom x) (denom y))))
(define (sub-rat x y)
(make-rat (- (* (numer x) (denom y))
(* (numer y) (denom x)))
(* (denom x) (denom y))))
(define (mul-rat x y)
(make-rat (* (numer x) (numer y))
(* (denom x) (denom y))))
(define (div-rat x y)
(make-rat (* (numer x) (denom y))
(* (denom x) (numer y))))
ASUS

Features

- Lexical scoping
- Macro system
- Large selection of builtin functions
- Simple test framework
- Tested:
  - 740 Lisp tests
  - 137 Rspec specs

Types

Symbols

Strings

Integers

Floats

Booleans

Lists

Association Lists

Functions

Primitives

Boxed Ruby Objects

Vectors

Frames

Primitives

Arithmetic
Conversions
Comparisons
Logical
Binary

Lists
Association Lists
Utility
Control structures
Assignment
Functions

Macros
Testing
Frames
Vectors
FFI

Uses

- application specific DSLs
- app functionality in Lisp
- extension language
- internal scripting/glue language
- DLC

Lisp from Ruby

Loading a File

```
result = parser.load_file(filepath)
```

- `filepath` names a file of lisp code
- parses and evaluates each s-expr in the file's contents
- returns the result of the last evaluation

Evaluating a String

```
result = parser.parse_and_eval_all(src)
```

- parses and evaluates each s-expr in src
- returns the result of the last evaluation

Example

```
code = """(define (double x) (+ x x))
          (double 5)"""

parser.parse_and_eval_all(code) ⇒ <10>
```

Evaluating a Single Sexpr

```
sexpr = parser.parse(src)
result = sexpr.evaluate(env)

result = parser.parse_and_eval(src, env)
```

Example

```
global = Lisp::EnvironmentFrame.global
parser = Lisp::Parser.new

f = parser.parse(“(define (double x) (+ x x))”)
f.evaluate(global)

expr = parser.parse(“(double 5)”)
expr.evaluate(global) ⇒ <10>
```

Example

```
parser = Lisp::Parser.new
parser.parse_and_eval(“(define (double x)(+ x x))”)
parser.parse_and_eval(“(double 5)”) ⇒ <10>
```

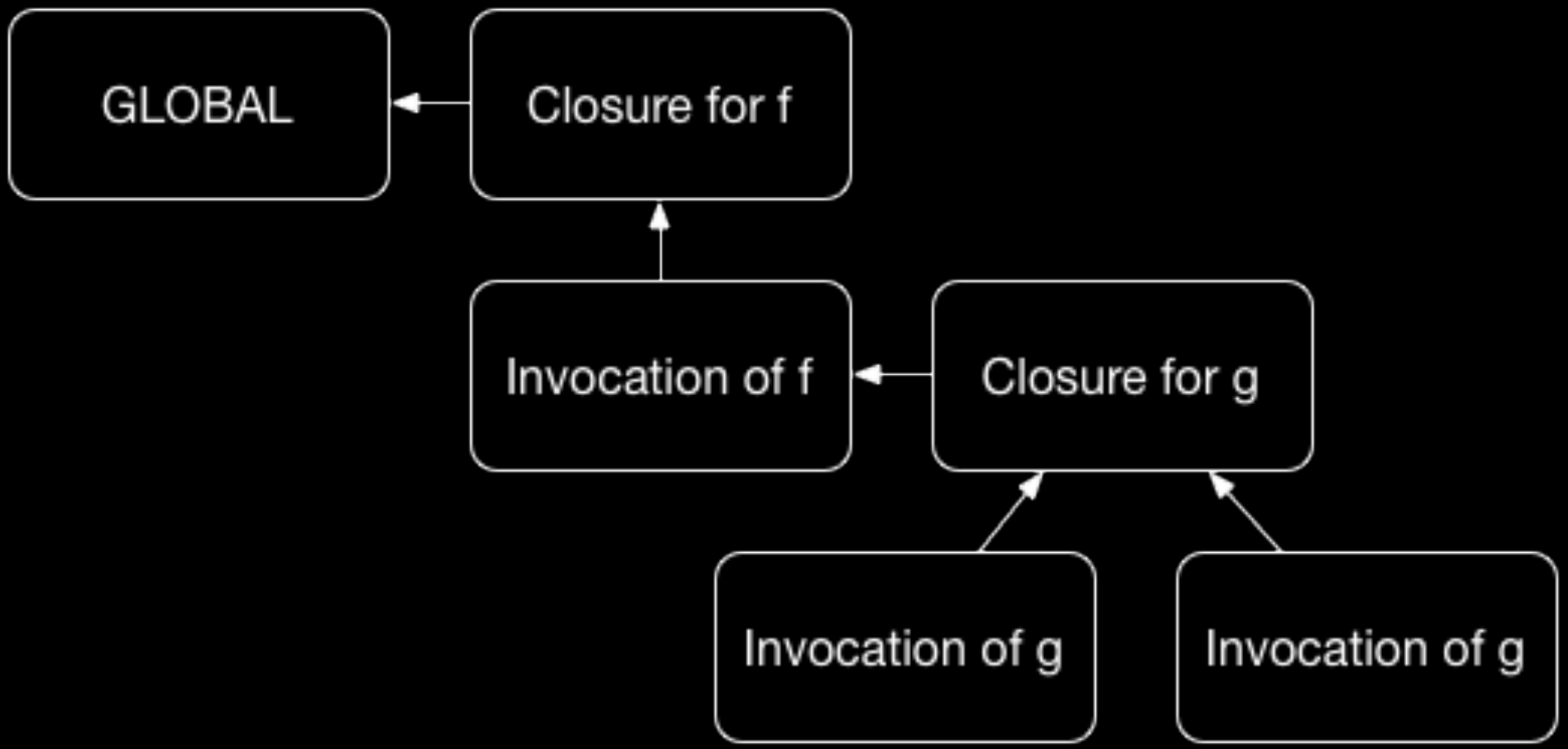
Lexical Scoping & Closures

```
(define x 42)

(define (f a)
  (define (g a)
    (if (eq? a 0)
        0
        (+ a (g (- a 1)))))

  (+ x (* a (g a))))

(f 1)
```


GLOBAL
Closure for f
Invocation of f
Closure for g
Invocation of g
Invocation of g

Apply

```
def evaluate_function(function_name)
  global = Lisp::EnvironmentFrame.global
  f = global.value_of(Lisp::Symbol.named(function_name))
  args = []
  f.arity.times do |i|
    arg = Lisp::Number.with_value(@stack.pop)
    args.unshift(arg)
  end
  arg_list = Lisp::ConsCell.array_to_list(args)
  result = f.apply_to(arg_list, global)
  @stack.push(result.value)
end
```

Ruby from Lisp

Instantiate and invoke

```
> (define h (Hash.))
<a Hash: {}>
> (.store h "a" 1)
1
> h
<a Hash: {"a"=>1}>
> (.include? h "a")
#t
> (.key h 1)
a
```

Extend

```
> (extend 'Object 'MyClass)
<a class: MyClass>
> (add-method 'MyClass 'foo (lambda () 42))
OK
> (define m (MyClass.))
<a MyClass: #<MyClass:0x007ff02425db10>>
> (.foo m)
42
```

Methods with args

```
> (extend 'Object 'MyClass)
<a class: MyClass>
> (add-method 'MyClass 'foo (lambda (x) (+ x 2)))
OK
> (define m (MyClass.))
<a MyClass: #<MyClass:0x007fbc1484a2d8>>
> (.foo m 5)
7
```

Primitives - Preamble

```
Lisp::Primitive.register("ruby-square",
                         “Return the square of
                          the argument",
                        true) do |args, env|
   ruby_square_impl(args, env)
end
```

Primitives - Signature

```
def ruby_square_impl(args, env)
```

Lisp list of arguments, and the evaluation environment

Returns the resulting Lisp object

Primitives - Implementing

```
def ruby_square_impl(args, env)
  raise “Expected 1 argument” if args.length != 1
  arg = args.car.evaluate(env)
  raise “Expected a number" unless arg.number?
  Number.with_value(arg.value * arg.value)
end
```

Primitives - Caveats

- Arguments are NOT pre-evaluated
  - If you need them evaluated, you need to do it
- Argument counts are are not checked
  - you have to do any arity checking
- Argument types are are not checked
  - you have to do it if needed

Opensource

Opensource

Standard MIT license

https://bitbucket.org/dastels/rubylisp

https://bitbucket.org/dastels/rubylisp-example

https://rubygems.org/gems/rubylisp

https://rubygems.org/gems/rubymotionlisp

http://daveastels.com/rubylisp

@dastels
dastels@acm.org