A World Without Assignment

Mountain West Ruby Conf
March 21, 2014
Aja Hammerly
@thagomizer_rb
http://github.com/thagomizer
http://www.thagomizer.com
SUBSTANTIAL
Functional Programming
What's That?
“In computer science, functional programming is a programming paradigm, a style of building the structure and elements of computer programs, that treats computation as the evaluation of mathematical functions and avoids state and mutable data.”
"The entire idea of mutable state is suspicious and easy to mess up"
“A functional language is just about calling functions.”

“No it's not about calling functions it is about creating functions that do things.”
This Talk
Setting Expectations

• 112 slides
• Lots of Code
• Lots of Parenthesis
• No Ponies
Why Should I Care?
Easier to Test
Concurrency
Safe Reuse
Brevity
You Already Use It
Ruby Makes It Easy
Scheme Basics
Prefix Notation

\((+ 5 3)\)
8
\((* 1 2 3)\)
6
\((+ (* 3 5) (- 10 6))\)
19
\((\text{add1} 6)\)
7

\(5 + 3\)
8
\(1 * 2 * 3\)
6
\((3 * 5) + (10 - 6)\)
19
\(6.\text{add1}\)
7
Functions

\[
\text{(define (square n) (\ast n n))}
\]

\[
\text{(square 5)}
\]

\[
25
\]

\[
\text{def square n}
\]

\[
\text{n * n}
\]

\[
\text{end}
\]

\[
\text{square 5}
\]

\[
25
\]
Conditionals

(define (abs x)
  (cond
   ((> x 0) x)
   ((= x 0) 0)
   (else (- x))))

def abs x
  case
  when x > 0
    x
  when x == 0
    0
  else
    x * -1
  end
end
**Conditionals**

```
(define (abs x)
  (cond
    ((> x 0) x)
    ((= x 0) 0)
    (else (- x))))
```

```python
def abs x
  case
    when x > 0
      x
    when x == 0
      0
    else
      x * -1
  end
end
```
Conditionals

(define (balmy t)
  (if (> t 65)
      #t
      #f))

def balmy? t
  if t > 65
    true
  else
    false
end
end
Lists

'(1 2 3)  [1, 2, 3]
Lists

'(1 2 3)
(car '(1 2 3))
  1

[1, 2, 3]
[1, 2, 3].first
  1
Lists

'(1 2 3)
(car '(1 2 3))
  1
(cdr '(1 2 3))
  '(2 3)

[1, 2, 3]
[1, 2, 3].first
  1
[1, 2, 3][1..-1]
  [2, 3]
Lists

'(1 2 3)
(car '(1 2 3))
1
(cdr '(1 2 3))
'(2 3)

[1, 2, 3]
[1, 2, 3].first
1
[1, 2, 3].rest
[2, 3]
Lists

'(1 2 3)
(car '(1 2 3))
  1
(cdr '(1 2 3))
  '(2 3)
(null? '())
  #t

[1, 2, 3]
[1, 2, 3].first
  1
[1, 2, 3].rest
  [2, 3]
[] .empty?
  true
Recursion
Factorial
(define (fact n)
  (if (= n 1)
      1
      (* n
         (fact (+ n 1))))
)

def fact n
  if n == 1
    1
  else
    n * fact(n - 1)
end
end
(define (fib n)
  (cond ((= n 0) 0)
        ((= n 1) 1)
        (else (+ (fib (- n 1))
                 (fib (- n 2))))))

def fib n
  case n
  when 0
    0
  when 1
    1
  else
    fib(n-1) + fib(n-2)
  end
end
Tail Call Optimization
Exponentiation

\[
\text{(define (expt b n)}
\begin{align*}
&\quad (\text{if} \ (= \ n \ 0) \\
&\quad \quad 1 \\
&\quad \quad (* \ b \\
&\quad \quad \ (\text{expt} \ b(- \ n \ 1))))
\end{align*}
\]

\[
\text{def expt(b, n)}
\begin{align*}
&\quad \text{if} \ n \ == \ 0 \\
&\quad \quad 1 \\
&\quad \text{else} \\
&\quad \quad b \ * \ \text{expt(b,n-1)}
\end{align*}
\]
end
expt(2, 4)
2 * expt(2, 3)
2 * 2 * expt(2, 2)
2 * 2 * 2 * expt(2, 1)
2 * 2 * 2 * 2 * expt(2, 0)
2 * 2 * 2 * 2 * 1
2 * 2 * 2 * 2
2 * 2 * 4
2 * 8
16
Tail Call Optimization

(define (expt b n)
 (expt-t b n 1))

(define (expt-t b c p)
  (if (= c 0)
   p
   (expt-t b
     (- c 1)
     (* b p)))))
\text{expt}(2, 4)
\quad \text{expt-t}(2, 4, 1)
\quad \text{expt-t}(2, 3, 2)
\quad \text{expt-t}(2, 2, 4)
\quad \text{expt-t}(2, 1, 8)
\quad \text{expt-t}(2, 0, 16)
\quad 16
Semi-Contrived
Example
Making Change
How many different ways can you make change of $1.00, given half-dollars, quarters, dimes, nickels, and pennies?
How many ways can you make some amount with some coins?
def count_change(amount, coins)
end
amount: number of cents
coins: a list of denominations
How many ways can you make some amount with some coins?
SIMPLIFY
How many ways can you make 1 cent using no coins?
def count_change(amount, coins):
    case when coins.empty?
        0
    end
end
> count_change(1, [])
0
alias cc count_change
> cc(1, [])
0
How many ways can you make 1 cent using pennies?
def cc(amount, coins)
    case
    when coins.empty?
        0
    when amount == coins.first
        1
    end
end
> cc(1, [1])

1
How many ways can you make 5 cents using pennies?
def cc(amount, coins)
    case
    when coins.empty?
        0
    when amount == coins.first
        1
    else
        cc(amount - coins.first, coins)
    end
end
> cc(5, [1])
1
How many ways can you make 5 cents using nickels and pennies?
> cc(5, [5, 1])

1
```python
def cc(amount, coins):
    case
    when coins.empty?
        0
    when amount == coins.first
        1
    else
        cc(amount - coins.first, coins)
    end
end
```
def cc(5, [5, 1])
    case
    when [5, 1].empty?
        0
    when 5 == 5
        1
    else
        cc(5 - 5, [5, 1])
    end
end
def cc(5, [5,1])
    case
    when [5,1].empty?
        0
    when 5 == 5
        1
    else
        cc(5 - 5, [5,1])
    end
end
def cc(5, [5,1])
    case
       when [5,1].empty?
           0
       when 5 == 5
           1
       else
           cc(5 - 5, [5,1])
    end
end
end
def cc(amount, coins):
    case
    when coins.empty?
        0
    when amount == coins.first
        1 + cc(amount, coins.rest)
    else
        cc(amount - coins.first, coins)
    end
end
> cc(5, [5, 1])
2
How many ways can you make 10 cents using nickels and pennies?
> cc(10, [5, 1])
2
def cc(10, [5,1])
  case
    when [5,1].empty?
      0
    when 10 == 5
      1 + cc(amount, coins.rest)
    else
      cc(10 - 5, [5,1])
  end
end
def cc(amount, coins):
    case
    when coins.empty?
        0
    when amount == 5
        1 + cc(amount, coins.rest)
    else
        cc(amount - 5, coins)
    end
end
def cc(10, [5,1])
    case
        when [5,1].empty?
            0
        when 10 == 5
            1 + cc(amount, coins.rest)
        else
            cc(10 - 5, [5,1])
    end
end
def cc(10, [5,1])
    case
        when [5,1].empty?
            0
        when 10 == 5
            1 + cc(amount, coins.rest)
        else
            cc(10 - 5, [5,1])
    end
end
def cc(amount, coins):
    case
    when coins.empty?
        0
    when amount == coins.first
        1 + cc(amount, coins.rest)
    else
        cc(amount, coins.rest) +
        cc(amount - coins.first, coins)
    end
end
> cc(10, [5, 1])
3
How many ways can you make 7 cents with nickels?
> cc(7, [5])
SystemStackError: stack level too deep
Uh-Oh
def cc(7, [5])
    case
    when [5].empty?
        0
    when 7 == [5].first
        1 + cc(7, [])
    else
        cc(7, []) +
            cc(7 - 5, [5])
    end
end
def cc(amount, coins):
    case
    when coins.empty?
        0
    when amount < coins.first
        cc(amount, coins.rest)
    when amount == coins.first
        1 + cc(amount, coins.rest)
    else
        cc(amount, coins.rest) +
            cc(amount - coins.first, coins)
    end
end
> cc(7, [5])
0
How many different ways can you make change of $1.00, given half-dollars, quarters, dimes, nickels, and pennies?
> cc(100, [50, 25, 10, 5, 1])
292
(define (cc amount coins)
  (cond
   ((null? coins) 0)
   ((< amount (car coins))
    (cc amount (cdr coins)))
   ((= amount (car coins))
    (+ 1
      (cc amount (cdr coins)))))
  (else
   (+
    (cc amount (cdr coins))
    (cc (- amount (car coins)) coins)))))
More Functions!
(define (member l n)
  (cond ((null? l)
         #f)
       ((= (car l) n)
        #t)
       (else
        (member (cdr l) n))))

def member(l, n)
  case
  when l.empty?
    false
  when l.first == n
    true
  else
    member(l.rest, n)
  end
end
Member

(define (member l n)
  (cond ((null? l)
         #f)
        ((= (car l) n)
         #t)
        (else
         (member (cdr l) n)))))

def member(l, n)
  case
  when l.empty?
    false
  when l.first == n
    true
  else
    member(l.rest, n)
  end
end
Any?

Lisp: (define (any l pred)
  (cond ((null? l) #f)
        ((pred (car l)) #t)
        (else (any (cdr l) pred)))))

Python: def any(l, pred):
    case when l.empty?
        false
    when pred.call(l.first)
        true
    else
        any(l.rest, pred)
    end
end
Anon. Functions

\[ ((\text{lambda} \ (x) \ (x \cdot x)) \ 3) \]

\[ \text{lambda} \ \{|x| \ x \ast x \}.\text{call}(3) \]

9

9
> (any '(1 2)
  (lambda (x)
    (< x 5)))
#t

> (any '(1 2)
  (lambda (x)
    (= x 5)))
#f

> any([1, 2],
    lambda { |x|
      x < 5})
true

> any([1, 2],
    lambda { |x|
      x == 5})
false
The Little Schemer

The Little Schemer
Fourth Edition

Daniel P. Friedman and Matthias Felleisen

Foreword by Gerald J. Sussman
The Little Schemer

What is \((-14\ 3)\)

What is \((-17\ 9)\)

What is \((-18\ 25)\)

Try to write the function \(-\)
List: Use subst

11.
8.
No answer. There are no negative numbers.

How about this:

```scheme
(define \(-\)
  (lambda (n m)
    (cond
      ((zero? m) n)
      (else (subst (+ n (subst m)))))))
```

1 I. S. This is like \(-\). Write it as \(-\) (see preface).

Can you describe how \((- n m)\) works?
It takes two numbers as arguments, and reduces the second until it hits zero. It subtracts one from the result as many times as it did to cause the second one to reach zero.

Is this a tup?
\((2\ 1\ 1\ 3\ 7\ 9\ 47\ 6)\)
Yes: tup is short for tuple.

Is this a tup?
\((8\ 5\ 5\ 5\ 5)\)
Yes, of course, it is also a list of numbers.

Is this a tup?
\((1\ 2\ 8\ apple\ 4\ 3)\)
No, it is just a list of atoms.

Is this a tup?
\((3\ (7\ 4)\ 13\ 9)\)
No, because it is not a list of numbers. (7 4) is not a number.
 Talks

• *Functional Programming and Ruby* by Pat Shaughnessy (GoRuCo 2013)

• *(Parenthetically Speaking)* by Jim Weirich (GoGaRuCo 2010)

• *Functional Principles for OO Development* by Jessica Kerr (Ruby Midwest 2013)

• *Y Not -- Adventures in Functional Programming* by Jim Weirich (Ruby Conf 2012)
Photo Credits

• http://cgeta.deviantart.com/art/Boring-Pinkie-Vector-205068021
Thank You