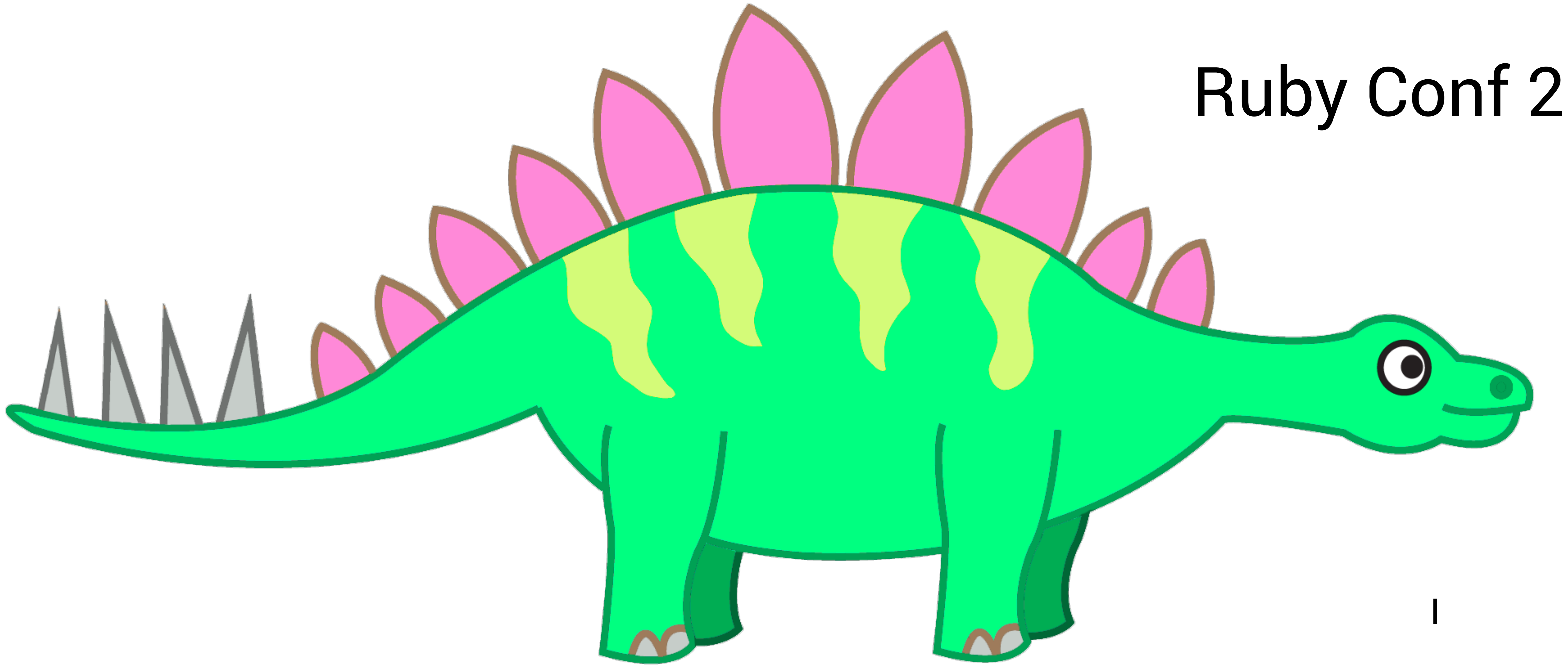
4 Programing Paradigms in 45 Minutes

Aja Hammerly (@the_thagomizer)

Ruby Conf 2017

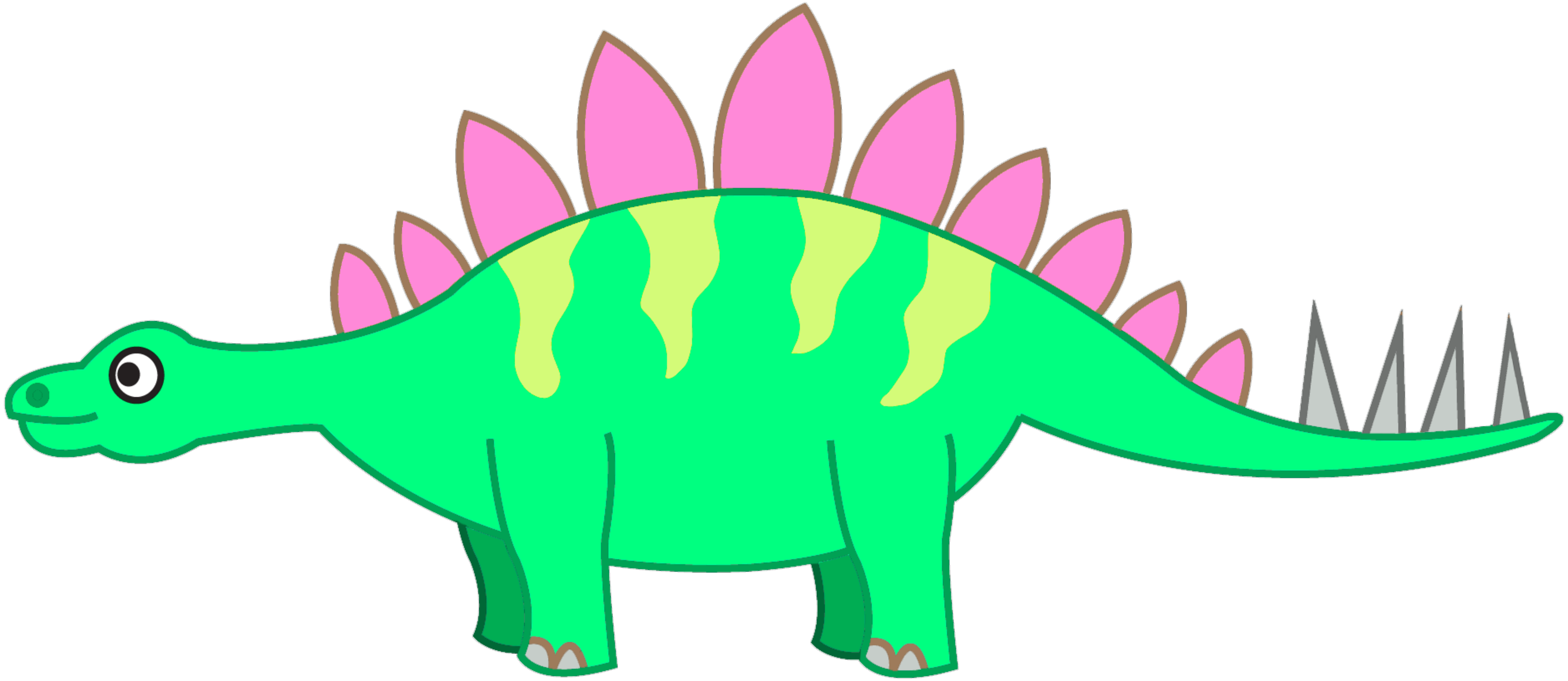


Aja Hammerly

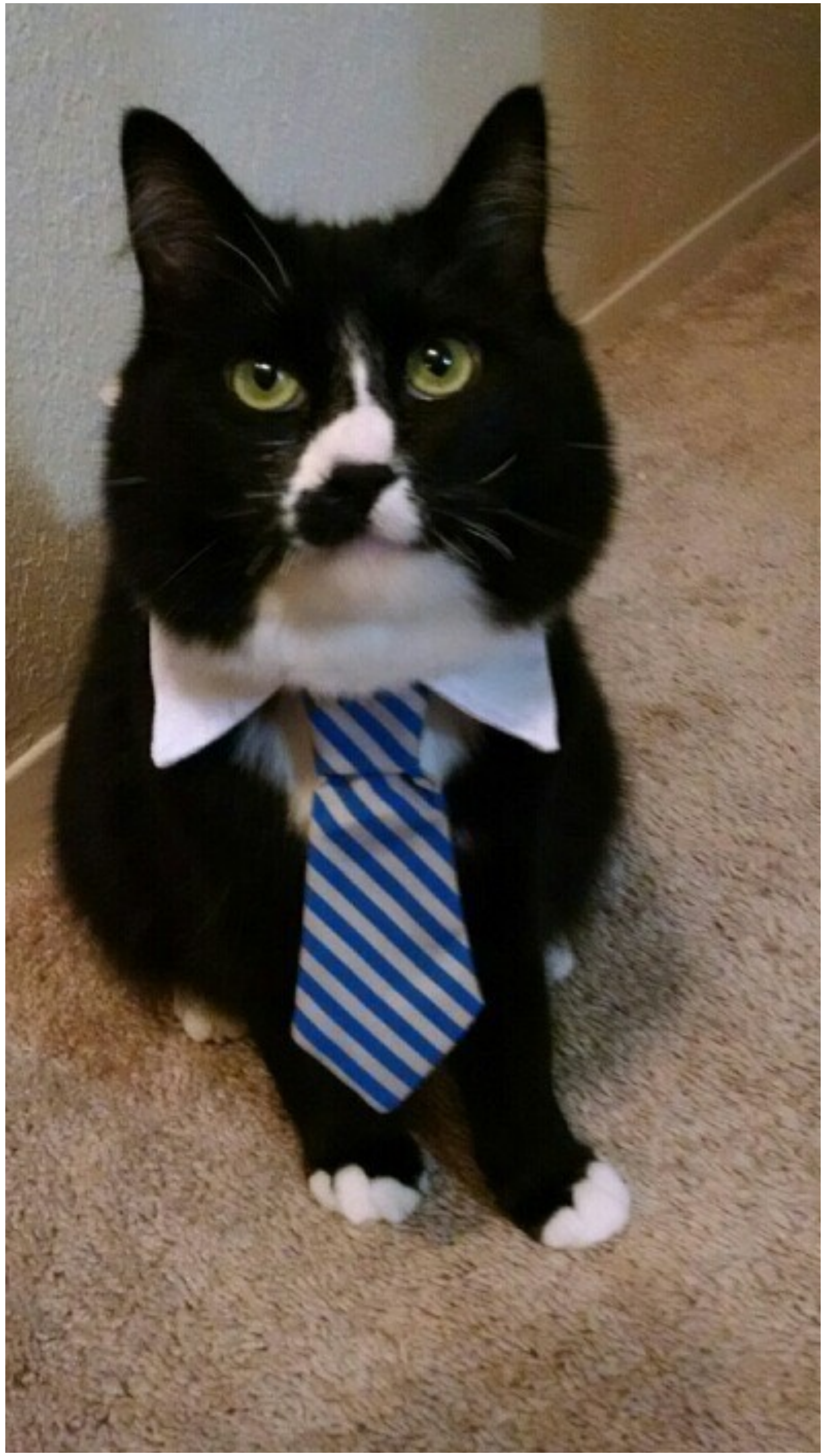
http://github.com/thagomizer

@the_thagomizer

http://www.thagomizer.com



Google Cloud Platform



Lawyer Cat Says:

Any code is copyright Google and licensed Apache V2

CS 60

Introduction to Principles of Computer Science

Abstraction

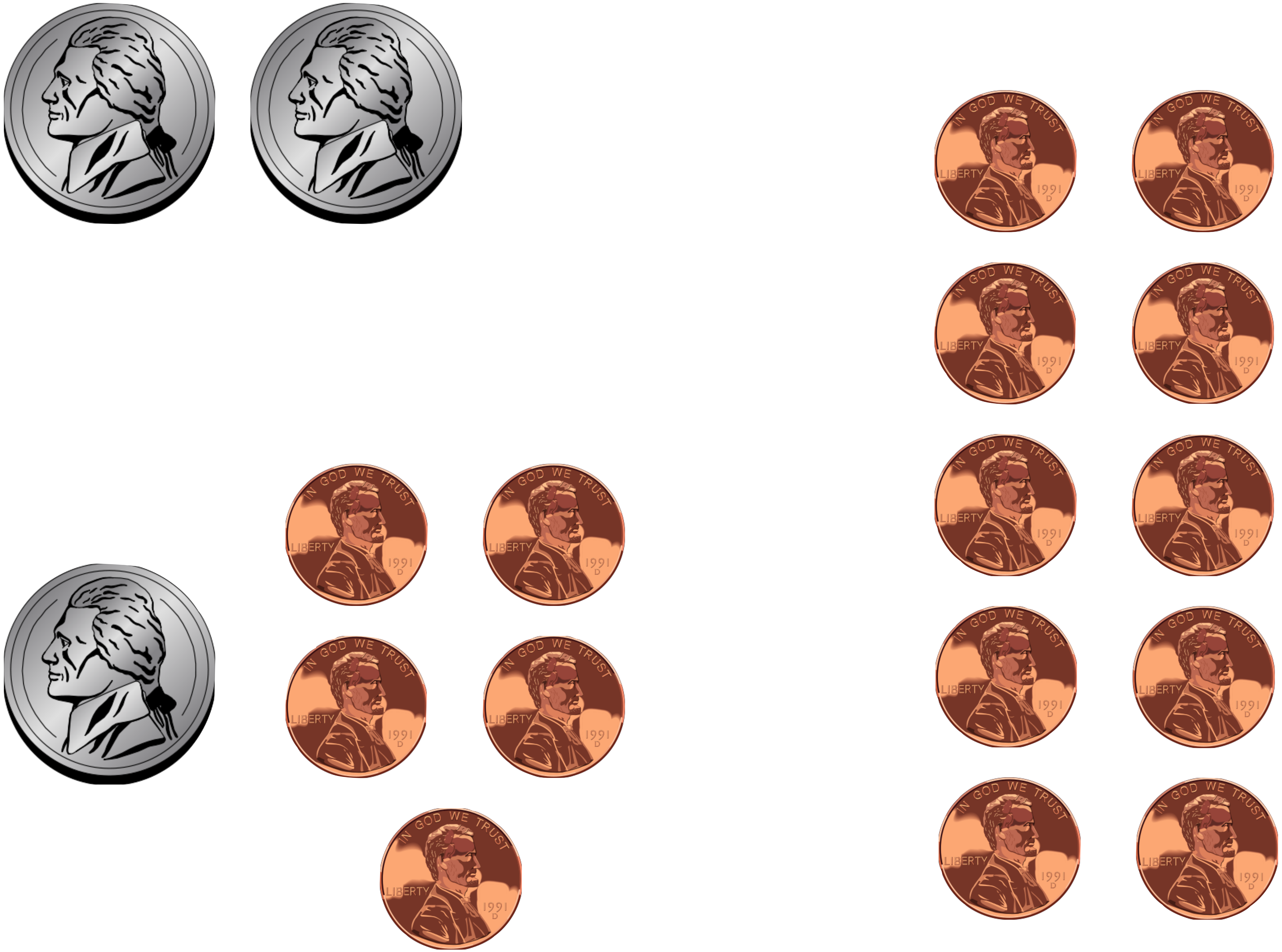
Polyglot

Similarities

Differences

Primary Example

Change


IN GOD WE TRUST
LIBERTY
1991
D

Object Oriented

Ruby

Overview

Everything Is An Object

State & Behavior

self

Objects Interact

Bank Account

```
class BankAccount

end
```

```
class BankAccount
  def initialize
    @balance = 0
  end
end
```

```
> account = BankAccount.new
```

```ruby
class BankAccount
  attr_reader :balance

  def initialize
    @balance = 0
  end
end
```

```
> account = BankAccount.new
#<BankAccount...>

> account.balance
0
```

```
class BankAccount
  attr_reader :balance

  def initialize
    @balance = 0
  end

  def deposit amount
    @balance += amount
  end

  def withdraw amount
    @balance -= amount
  end
end
```

```
> account = BankAccount.new
#<BankAccount...>

> account.balance
0

> account.deposit 100
> account.withdraw 30

> account.balance
70
```

```
class BankAccount
  attr_reader :balance

  def initialize
    @balance = 0
  end

  def deposit amount
    @balance += amount
  end

  def withdraw amount
    @balance -= amount
  end
end
```

Strengths

Modeling

Reusability

Ease of Testing

Making Change

```ruby
class CashRegister
  attr_reader :drawer

  def initialize
    @drawer = [2000, 1000, 500, 100,
               25, 10, 5, 1]
  end

  def make_change bill, tendered
    difference = tendered - bill

    change = []
    i = 0
    denomination = @drawer[i]

    while difference > 0 do
      if difference < denomination
        i += 1
        denomination = @drawer[i]
        next
      end

      change << denomination
      difference -= denomination
    end

    change
  end
end
```

```
class CashRegister
  attr_reader :drawer

  def initialize
    @drawer = [2000, 1000,
               500,  100,
               25,   10,
               5,    1]
  end
end
```

```
def make_change owed, tendered
  difference = tendered - owed

  change = []
  i = 0
  denomination = @drawer[i]

  while difference > 0 do
    if difference < denomination
      i += 1
      denomination = @drawer[i]
      next
    end

    change << denomination
    difference -= denomination
  end

  change
end
```

```
def make_change owed, tendered
  ...
end
```

```ruby
 def make_change owed, tendered
   difference = tendered - owed

   change = []
end
```

```
def make_change owed, tendered
  difference = tendered – owed

  change = []
  i = 0
  denomination = @drawer[i]
end
```

```
def make_change owed, tendered
  difference = tendered - owed

  change = []
  i = 0
  denomination = @drawer[i]

  while difference > 0 do
    ...
  end
end
```

```
while difference > 0 do
  if difference < denomination
    i += 1
    denomination = @drawer[i]
    next
  end

  change << denomination
  difference –= denomination
end
```

```
while difference > 0 do
  if difference < denomination
    i += 1
    denomination = @drawer[i]
    next
  end

  change << denomination
  difference –= denomination
end
```

```
class CashRegister
  attr_reader :drawer

  def initialize
    @drawer = [2000, 1000, 500, 100,
               25, 10, 5, 1]
  end

  def make_change bill, tendered
    difference = tendered - bill

    change = []
    i = 0
    denomination = @drawer[i]

    while difference > 0 do
      if difference < denomination
        i += 1
        denomination = @drawer[i]
        next
      end

      change << denomination
      difference -= denomination
    end

    change
  end
end
```

Functional

Racket

Overview

Functions

Pure Functional

Input -> Output

Data

Procedures

Syntax

Infix vs. Prefix

```
(+ 3 5)
8

(* 1 2 3)
6

(+ (* 3 5)
   (- 10 6))
19
```

Functions

```
(define (square n)
  (* n n))


(square 5)
  25
```

Conditionals

```
(cond
  ((test) stuff if test is true)
  ((different test) different stuff)
  (else more stuff))
```

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

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

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

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

Lists

```
'(1 2 3)
```

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

(cdr '(1 2 3))
'(2 3)
```

```
(cons '1 '(2 3))
'(1 2 3)
```

Examples

Factorial

```
(define (fact n)
  (cond
     ((<= n 1)
       1)
     (else
      (* n (fact (- n 1))))))
```

Fibonacci

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

Strengths

Concurrency

Easier To Test

Reusability

Brevity

Making Change

```
(define (make–change x denoms)
  (cond
    ((= x 0)
     '())
    ((empty? denoms)
     false)
    ((< x (car denoms))
     (make–change x (cdr denoms)))
    (else
     (cons (car denoms) (make–change (– x
(car denoms)) denoms)))))
```

```
(define (make–change x denoms)
  (cond
    ((= x 0)
     '())
    ((empty? denoms)
     false)
    ((< x (car denoms))
     (make–change x (cdr denoms)))
    (else
     (cons (car denoms) (make–change (– x
(car denoms)) denoms)))))
```

```
(define (make-change x denoms)
  (cond
    ((= x 0)
     '())
    ((empty? denoms)
     false)
    ((< x (car denoms))
     (make-change x (cdr denoms)))
    (else
     (cons (car denoms) (make-change (- x
(car denoms)) denoms)))))
```

```
(define (make-change x denoms)
  (cond
    ((= x 0)
     '())
    ((empty? denoms)
     false)
    ((< x (car denoms))
     (make-change x (cdr denoms)))
    (else
     (cons (car denoms) (make-change (- x
(car denoms)) denoms)))))
```

```
(define (make-change x denoms)
  (cond
    ((= x 0)
     '())
    ((empty? denoms)
     false)
    ((< x (car denoms))
     (make-change x (cdr denoms)))
    (else
     (cons (car denoms) (make-change (- x
(car denoms)) denoms)))))
```

```
(define (make-change x denoms)
  (cond
    ((= x 0)
     '())
    ((empty? denoms)
     false)
    ((< x (car denoms))
     (make-change x (cdr denoms)))
    (else
     (cons (car denoms)
           (make-change (- x (car denoms))
denoms))))))
```

Logic/Constraint

Prolog

Overview

Formal Logic

Facts & Clauses

What NOT How

Syntax

```
VARIABLE

constant
```

```
state(washington).
border(washington, oregon).
border(washington, idaho).
border(oregon, california).
```

```
adjacent(X, Y) :- border(X, Y).
```

Pattern Matching

```
adjacent(X, Y) :- border(X, Y).
```

```
border(washington, oregon).
border(washington, idaho).

adjacent(X, Y) :- border(X, Y).

?- adjacent(washington, oregon).
yes

?- adjacent(oregon, washington).
no
```

```
adjacent(X, Y) :- border(X, Y).
adjacent(X, Y) :- border(Y, X).
```

Basic Examples

Ancestors

```
father(homer, bart).
father(homer, lisa).
mother(marge, bart).
mother(marge, lisa).
```

```
?- mother(X, bart).
X = marge

?- mother(marge, Y).
Y = bart ? ;
Y = lisa
```

```
sibling(X, Y) :-
  mother(Z, X),
  mother(Z, Y),
  X \== Y.

sibling(X, Y) :-
  father(Z, X),
  father(Z, Y),
  X \== Y.
```

```
?- sibling(X, Y).
X = bart
Y = lisa
```

Lists

```
[]
[1, 2, 3]
[apples, bananas]
[apples, [1, 3], mangos]
```

[F | R]

```
[1, 2, 3]

[F | R]
F = 1
R = [2, 3]
```

–

Member

```
member(X, [X | _]).
member(X, [_ | R]) :- member(X, R).
member(X, [X | _]).
member(X, [_ | R]) :- member(X, R).
member(X, [X | _]).
member(X, [_ | R) :- member(X, R).
```

Strengths

Flexibility

Constraints

Making Change

```
change(amount, coins,
change)
```

```
change(0, _, []).
change(A, [F | R], [F | X]) :-
       A >= F,
       B is A - F,
       change(B, [F | R], X).
change(A, [_ | R], X) :-
       A > 0,
       change(A, R, X).
change(0, _, []).
change(A, [F | R], [F | X]) :-
       A >= F,
       B is A - F,
       change(B, [F | R], X).
change(A, [_ | R], X) :-
       A > 0,
       change(A, R, X).
change(0, _, []).
change(A, [F | R], [F | X]) :-
       A >= F,
       B is A - F,
       change(B, [F | R], X).
change(A, [_ | R], X) :-
       A > 0,
       change(A, R, X).
change(0, _, []).
change(A, [F | R], [F | X]) :-
       A >= F,
       B is A - F,
       change(B, [F | R], X).
change(A, [_ | R], X) :-
       A > 0,
       change(A, R, X).
change(0, _, []).
change(A, [F | R], [F | X]) :-
       A >= F,
       B is A - F,
       change(B, [F | R], X).
change(A, [_ | R], X) :-
       A > 0,
       change(A, R, X).
change(0, _, []).
change(A, [F | R], [F | X]) :-
       A >= F,
       B is A - F,
       change(B, [F | R], X).
change(A, [_ | R], X) :-
       A > 0,
       change(A, R, X).
change(0, _, []).
change(A, [F | R], [F | X]) :-
       A >= F,
       B is A - F,
       change(B, [F | R], X).
change(A, [_ | R], X) :-
       A > 0,
       change(A, R, X).
change(0, _, []).
change(A, [F | R], [F | X]) :-
       A >= F,
       B is A - F,
       change(B, [F | R], X).
change(A, [_ | R], X) :-
       A > 0,
       change(A, R, X).
change(0, _, []).
change(A, [F | R], [F | X]) :-
       A >= F,
       B is A - F,
       change(B, [F | R], X).
change(A, [_ | R], X) :-
       A > 0,
       change(A, R, X).
```

Owww! My Brain.

Procedural

Assembly

Overview

Syntax

Registers

A

M

Computations

A + D

D - A

A - D

[A or D] + 1

[A or D] - 1

! & |

- [A or D]

$M + 1$

D + M

Assignment

$$D = M + 1$$

$$D = D - A$$

$$MD = A + 1$$

@Integer

@100

@Label

Jumps

val;jump type

D;JGT

JEQ

0;JEQ

JGT, JLT, JGE, JLE

Basic Examples

Add

```
@2
D=A
@3
D=D+A
@0
M=D
```

```
@2
D=A
@3
D=D+A
@0
M=D
```

```
@2
D=A
@3
D=D+A
@0
M=D
```

```
@2
D=A
@3
D=D+A
@0
M=D
```

```
@2
D=A
@3
D=D+A
@0
M=D
```

```
@2
D=A
@3
D=D+A
@0
M=D
```

Sum

```
@0
M=0
@5
D=A
@1
M=D

(LOOP)
@1
D=M
@0
M=M+D

@1
MD=M-1

@END
D;JLE
@LOOP
0;JMP

(END)
@END
0;JMP
```

```
@0
M=0
@5
D=A
@1
M=D
```

```
(LOOP)
@1
D=M
@0
M=M+D
```

```
@1
MD=M-1
```

```
@END
D;JLE
@LOOP
0;JMP
```

Strengths

Strengths?

Simple

Scripting

Easy to Write

Making Change

```
@67
D=A
@R0
M=D

// Load Denominations
@25
D=A
@R1
M=D

@10
D=A
@R2
M=D

@5
D=A
@R3
M=D
@1
D=A
@R4
M=D

(QUARTERS)
@R0
D=M
@R1
D=D-M
@DIMES
D;JLT
@R0
M=D
@R5
M=M+1
@QUARTERS
0;JMP

(DIMES)
@R0
D=M
@R2
D=D-M
@NICKELS
D;JLT
@R0
M=D
@R6
M=M+1
@DIMES
0;JMP

(NICKELS)
@R0
D=M
@R3
D=D-M
@PENNIES
D;JLT
@R0
M=D
@R7
M=M+1
@NICKELS
0;JMP


(PENNIES)
@R0
D=M
@R4
D=D-M
@END
D;JLT
@R0
M=D
@R8
M=M+1
@PENNIES
0;JMP

(END)
@END
0;JMP
```

```
M0:       Amount to make
M1 – M4: Coin denominations
M5 – M8: Number of each coin to use
```

```
R0:         Amount to make
R1 – R4: Coin denominations
R5 – R8: Number of each coin to use
```

```
@67
D=A
@R0
M=D
```

```
@25
D=A
@R1
M=D

@10
D=A
@R2
M=D
```

```
@5
D=A
@R3
M=D

@1
D=A
@R4
M=D
```

```
(QUARTERS)
@R0
D=M
@R1
D=D-M
@DIMES
D;JLT
@R0
M=D
@R5
M=M+1
@QUARTERS
0;JMP
```

Learn More

Functional

Talks

(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)

Books

Friedman, Daniel & Felleisen, Matthias. The Little Schemer.

Abelson, Harold et al. Structure and Interpretation of Computer Programs

Logic

Talks

- A Taste of Prolog by Aja Hammerly (Cascadia Ruby 2012)

Books

- Sterling, Leon & Shapiro, Ehud. The Art of Prolog
- Clocksin, William F. Clause and Effect: Prolog Programming for the Working Programmer
- Bratko, Ivan. Prolog Programming for Artificial Intelligence

Procedural

Books

Nisan, Noam & Schocken, Shimon. The Elements of Computing Systems: Building a Modern Computer from First Principles

General

Books

Tate, Bruce A. Seven Languages in Seven Weeks: A Pragmatic Guide to Learning Programming Languages

Lopes, Cristina Videira. Exercises in Programming Style

Thoughtful Closing

Thank You