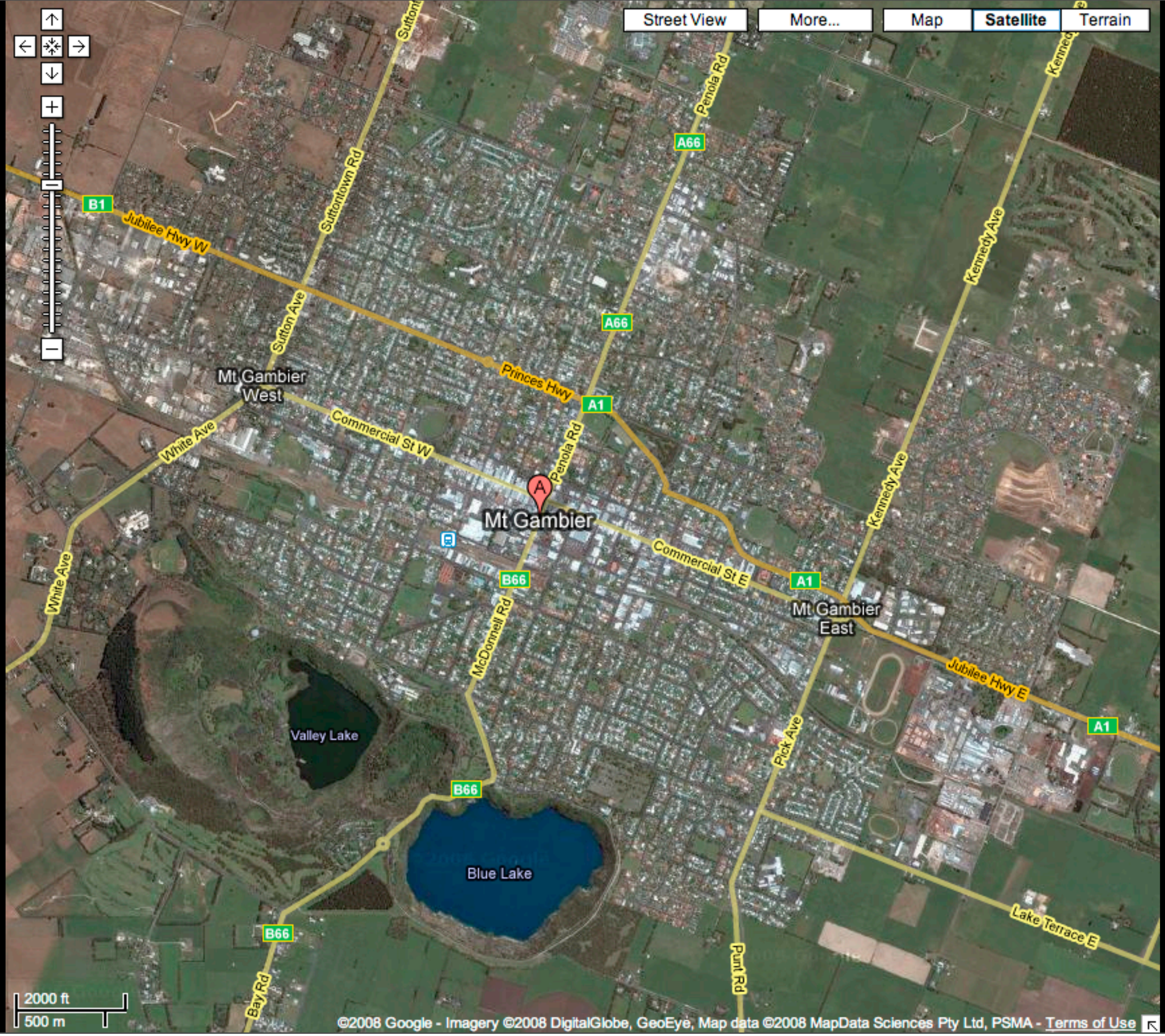
AUC Ruby Course 2009

Marcus Crafter & Gareth Townsend

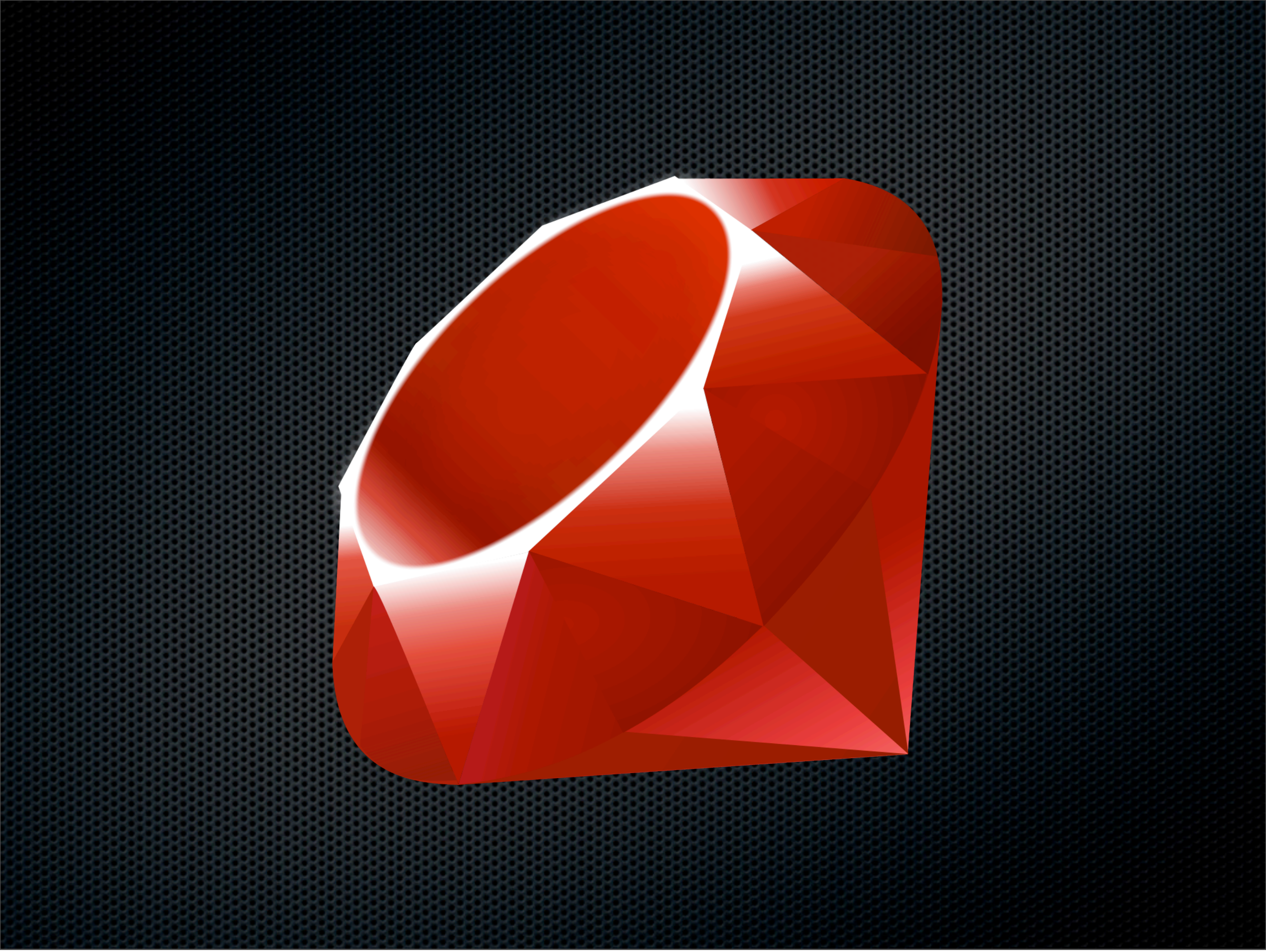
Marcus Crafter

crafterm@redartisan.com
http://twitter.com/crafterm


Street View
More...
Map
Satellite
Terrain
B1
Jubilee Hwy W
Suttontown Rd
Sutton Ave
Penola Rd
A66
Kennedy Ave
Princes Hwy
A1
Mt Gambier West
Commercial St W
White Ave
Mt Gambier
Commercial St E
Mt Gambier East
Jubilee Hwy E
B66
McDonnell Rd
Valley Lake
Blue Lake
Pick Ave
Lake Terrace E
Punt Rd
Bay Rd
2000 ft
500 m
©2008 Google - Imagery ©2008 DigitalGlobe, GeoEye, Map data ©2008 MapData Sciences Pty Ltd, PSMA - Terms of Use


LA TROBE
UNIVERSITY

Wednesday, 11 March 2009



red() artisan

crafterm@redartisan.com

Be...

- Be Brave!
- Be Excited!
- Be Afraid!
- Be Challenged!
- Be Sober! :)

Gareth Townsend

Software Developer

Background

- From Melbourne
- Lived in Papua New Guinea and Vienna
- Studied Software Engineering at RMIT
- Founded Melbourne CocoaHeads

Work Experience

- 3 years professional Software Engineering
- Ruby on Rails, some .NET and CocoaTouch
- AUC training courses and Dev World

Work Experience

- Mac user since 2003
- Ruby and Rails since 2005
- Ruby and Rails professionally since 2007

Work Experience

- Hannan IT
- Six Figures
- Box + Dice
- CLEAR

Conferences

- WWDC (twice)
- Web Directions South
- Dev World (speaker)

Lets start with a Demo!

What you can do with Ruby

Archaeopteryx

MIDI control device for DJ'ing

CLEAR

Australia's Grain Exchange

Twitter

Instant online communication


US AIRWAYS

yellowlab.com.au

Yellow Pages Testing Ground

The Ruby Value

The Ruby Value

- Competitive Edge
  - Increased development speed
  - Smaller development teams, lower costs
- Exciting Companies
  - Early adopters, startups, corporations
- Ruby on Rails has been a big catalyst for Ruby
- Large community conferences, books, user groups, etc

Course Agenda

General Overview

General Overview

- Ruby, the language and platform
- Assume no prior Ruby knowledge and start from the beginning
  - Ruby is an easy language to get started with, but has plenty of depth to master
    - Today, language, environment, and get started writing code
    - Tomorrow, advanced topics, meta prog & DSLs

General Overview

- RubyCocoa
  - Ruby integration with Mac OS X/Cocoa
- Ruby on Rails
  - Web framework, catalyst for Ruby's popularity
  - AddressBook and Database integration

General Overview

- We'll also be building things along the way
  - Twitter applications, command line & RubyCocoa
  - Rails, Address Book manipulation application
- Time to experiment, questions and answers

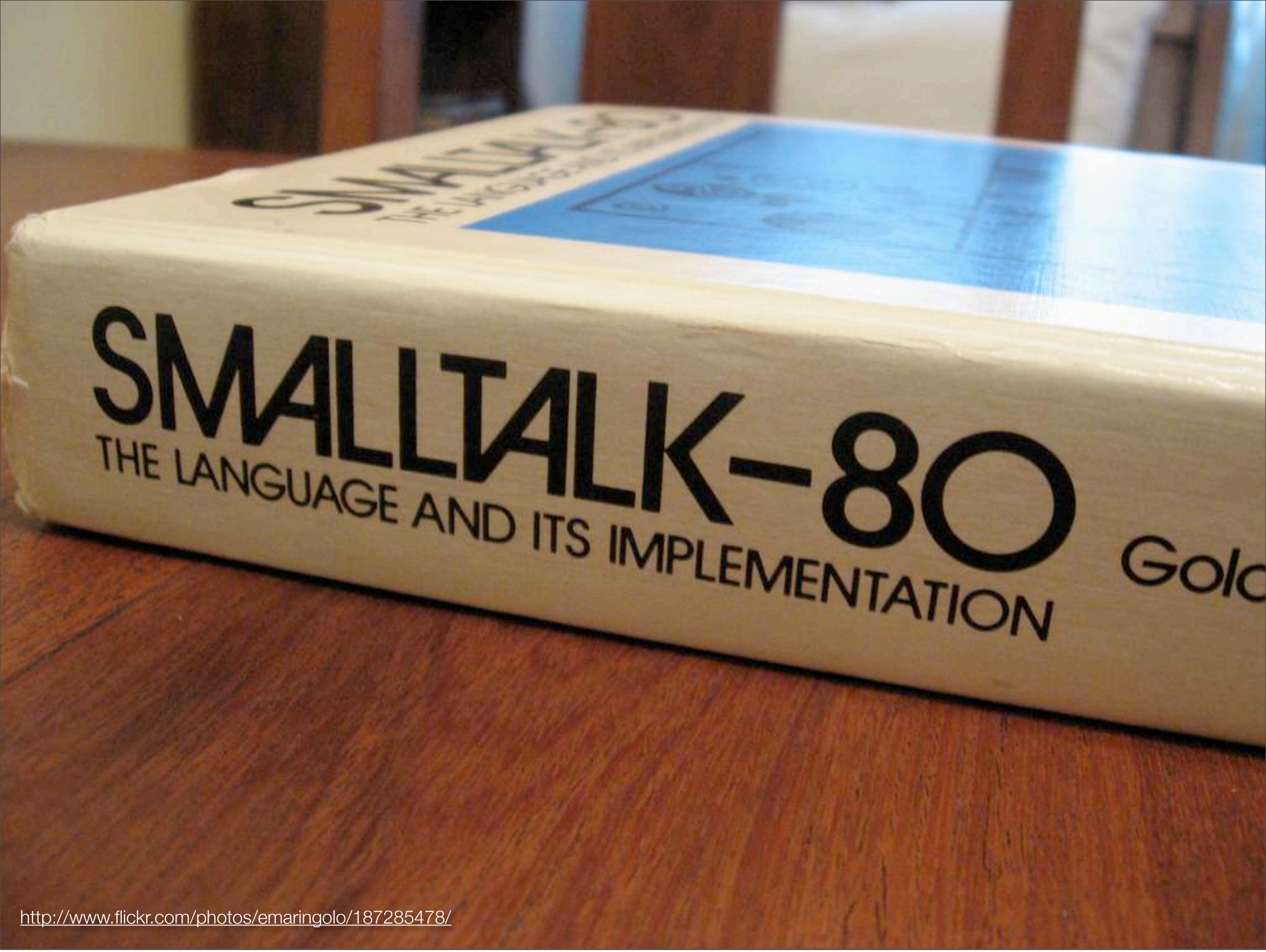
So what is Ruby?

"Ruby is... a dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write"

http://flickr.com/photos/76128093@N00/542486031

What is Ruby?

- Interpreted, dynamic language
  - No separate compilation or build phase
- Dynamically typed
  - No declaration of types when you reference objects
  - No compile time type checking
    - All done at runtime
- Flexibility and power, very interesting things! :)



http://www.flickr.com/photos/emaringolo/187285478/

What is Ruby?

- Blend of Matz's favourite languages such as Perl, Smalltalk, Eiffel, Ada, Lisp, and others
- Open Source
  - Free, as in speech, not beer
- Dual Licensed under the GPL or the Ruby license
  - Can be used commercially, modified, and distributed

What is Ruby?

- Object Oriented - absolutely everything is an object
  - Including "*fundamental*" types

```
5.times { print "We *love* Ruby!" }
```

- Message based
  - Object#send(:symbol) is the identical to a method invocation

What is Ruby?

- Dynamic runtime
  - Define classes and methods at runtime, on the fly
- Can create new domain specific languages, eg:
  - Software construction (rake) or testing (rspec)
  - Software deployment (sprinkle)
  - Database manipulation (active record)

Rake - Construction

```
desc "Source installer task"
task :install => :environment do |t|
    FileUtils.mkdir_p @target
    FileUtils.cp_r @source, @target
end
```

```
$> rake install
```

RSpec - BDD Testing

```
describe MyClass, ‘when created’ do
    it ‘should be invalid without a name’ do
        @instance.should validate_presence_of(:name)
    end
end
```

```
$> spec spec/models/myclass_spec.rb
```

Sprinkle - Deployment

```
package :ruby do
    description 'Ruby Virtual Machine'
    version '1.8.7'
    source "ftp://ftp.ruby..org-#{version}-p111.tar.gz"
    requires :ruby_dependencies
end
```

```
$> sprinkle -c -s deployment.rb
```

Databases

```
class User < ActiveRecord::Base
    validates_presence_of :name
    validates_uniqueness_of :name
    validates_numericality_of :age, :greater_than => 0
end
```

```
User.find :all, :conditions => { :name => 'Steve' }
```

All examples were pure Ruby

Ruby Flavours

MRI, Rubinius, MacRuby, JRuby & IronRuby

MRI - “Matz” Ruby

- Official source tree
- 1.8.x stable release, 1.9.x in development
- Interpreter, language and runtime written in C
- Standard library written in Ruby
- Interesting legacy and history
- 1.8.x green threads, 1.9.x native threads

Rubinius - Smalltalk inspired

- Completely new Ruby 1.8 VM written based on the Smalltalk Blue Book architecture
- Small fast and tight VM written in C++
- Language, standard library and environment written in Ruby
- Created the Ruby Test project to define what the Ruby language is
- Can run a test Rails application

IronRuby

- Ruby 1.8 implementation based on Microsoft's Dynamic Language Runtime
- Sponsored by Microsoft
- Still in active development
- http://www.ironruby.net/

JRuby - Java Ruby

- First Ruby 1.8 port to a new environment
- First Ruby environment to offer native threads via JVM
- Runs Rails in production
  - Deployment of Rails application as WAR files
- Competitive MRI performance

MacRuby

- Ruby 1.9 implementation written with Objective-C and CoreFoundation/Cocoa
- Very exciting, Ruby types are Cocoa types
- Cocoa garbage collection
- Sponsored by Apple
- In active development
- http://www.macruby.org

Ruby Environment

/usr/bin/ruby, /usr/bin/irb, /usr/bin/ri, gems

/usr/bin/ruby

- Main Ruby binary
- Runs your Ruby script
- Generally named in the *she-bang* path of scripts

  #!/usr/bin/env ruby

/usr/bin/irb

- Interactive ruby console
- Allows you to write Ruby code directly into the interpreter
- Very useful for testing
- Provides syntax and command completion

/usr/bin/ri & /usr/bin/rdoc

- Inbuilt documentation system
- Similar to manpages for Ruby methods, classes, etc

Gems

- Ruby library deployment system
  - Packaging
  - Dependencies
  - Networking, remote installation

Ruby First Glance

Straight in the deep end!

Hello World

```
class Hello
  def self.say(rant)
    puts "hello #{rant}"
  end
end

Hello.say('world') # => "hello world"
```

Some Comparisons?

Ruby

```ruby
require 'book'
require 'glossy'
require 'softcover'

class Magazine < Book
  include Glossy, SoftCover

  attr_accessor :title, :content

  def initialize(title, content)
    @title = title
    @content = content
  end

  def to_s
    "#{@title}: #{@content}"
  end

end
```

Objective-C

```objc
#import "book.h"
#import "glossy.h"
#import "softcover.h"

@interface Magazine : Book <Glossy, SoftCover> {
  NSString * title;
  NSString * content;
}

@property (nonatomic, assign) NSString * title;
@property (nonatomic, assign) NSString * content;

- (id)initWithTitle:(NSString *)title andContent:(NSString *)content;

@end
```

```objc
@implementation Magazine
@synthesize title, content;

- (id)initWithTitle:(NSString *)aTitle andContent:(NSString *)someContent {
    if (self = [super init]) {
      title = aTitle;
      content = someContent;
    }
    return self;
}

- (NSString *)description {
  return [NSString stringWithFormat:@"%s: %s", title, content];
}

@end
```

Java

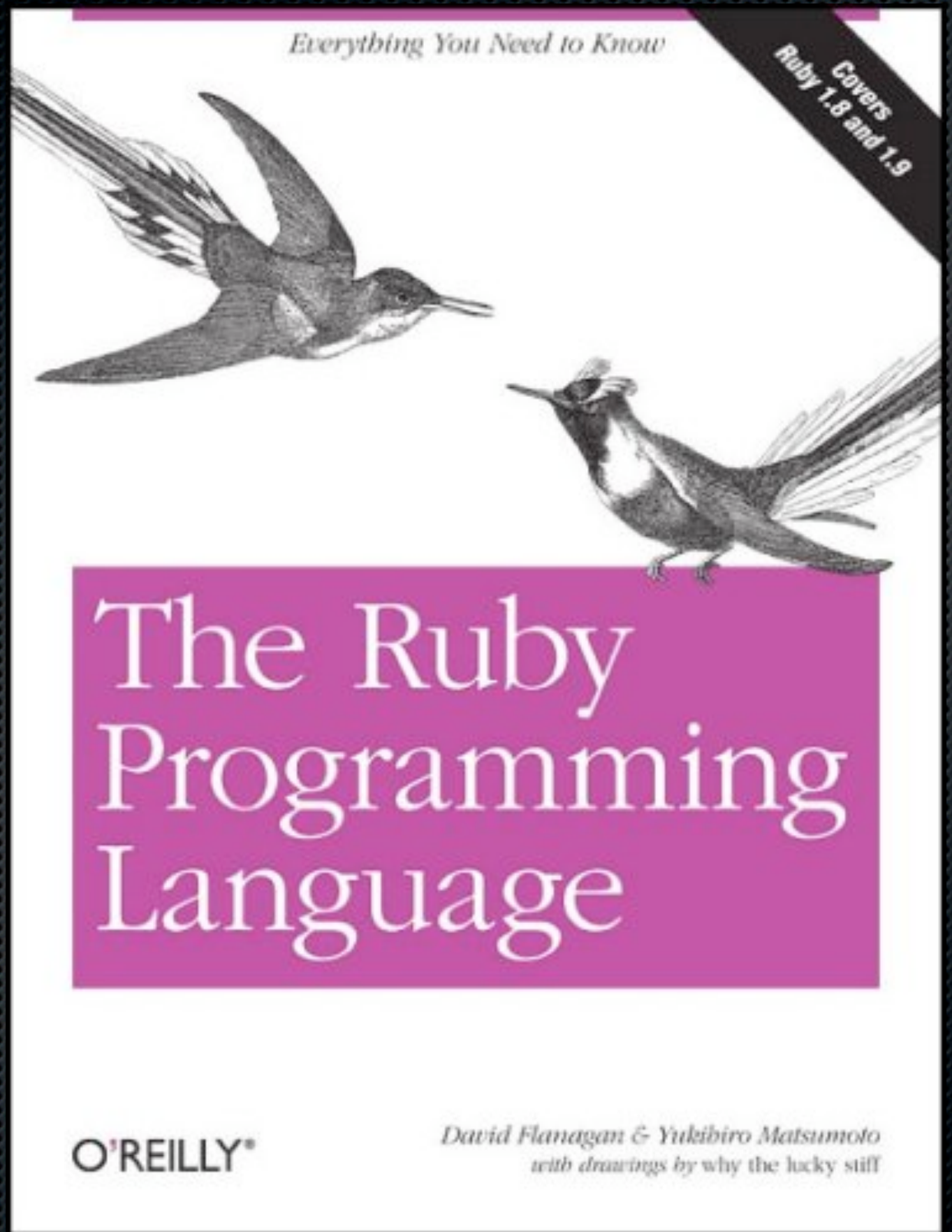
```java
import com.company.books.Book;
import com.company.books.printing.styles.Glossy;
import com.company.books.covers.SoftCover;

public class Magazine extends Book implements Glossy, SoftCover {

  public Magazine(String title, String content) {
    this.title = title;
    this.content = content;
  }

  public void setTitle(String title) {
    this.title = title;
  }

  public String getTitle() {
    return title;
  }

  // ... accessors/getters for String content ...

  public String toString() {
    return title + ": " + content;
  }

  private String title;
  private String content;
}
```

JavaScript

Ruby The Language

The Details!

Reference Material


Everything You Need to Know
Covers Ruby 1.8 and 1.9
The Ruby Programming Language
O'REILLY®
David Flanagan & Yukihiro Matsumoto
with drawings by why the lucky stiff

The Pragmatic Programmers

RubyCocoa

Bringing Some Ruby Love to OS X Programming

Brian Marick

Edited by Daniel H. Steinberg

The Facets of Ruby Series

The Pragmatic Programmers

Covers Rails 2

Agile Web Development with Rails

Third Edition

Sam Ruby

Dave Thomas

David Heinemeier Hansson

with Leon Breedt, Mike Clark, Justin Gehtland, James Duncan Davidson, and Andreas Schwarz

RAILS

The Facets of Ruby Series

Data Types

Numbers, Text, Literals, Collections, etc

Numbers

- Represented by Objects

```
10.times do |i|
  print "#{i}"
end # => 0123456789

5.to_s # => String “5”

3.send :+, 5 # => 3 + 5 = 8
```

- Numeric, base class for all numbers

Integers

- Integer, base class for all integers
- Fixnum, if it fits within 31 bits

```
5.class # => Fixnum
```

- Bignum, if larger

```
1234567890.class # => Bignum
```

- Fixnum to Bignum converted automatically

Floats

- Floating point numbers
  - IEEE-754 specification
    - Binary representation for efficiency
      - Not precise (example)
- BigDecimal class
  - Decimal representation, unlimited size
  - Financial applications, etc

Text

- Represented by the String class
- Mutable sequences of characters
  - Ruby 1.9 adds comprehensive multi-byte support
- Quoting
  - Single, double, %q/Q and HERE documents

Single quoted Strings

```
'the quick brown fox jumps over the lazy dog'
```

- Not escaped
- No variable interpolation
- No special characters such as \

Double quoted Strings

```
“the #{speed} brown #{animal} jumps over the
#{mood} dog\n”
```

- Escape sequences
- Variable interpolation

%q/Q quoted Strings

```
%q(the ‘quick’ brown fix jumps over the ‘lazy’
dog)

%Q(the “quick” #{colour} fix jumps over the
‘lazy’ dog)
```

- %q equivalent to a single quoted string, including ‘
- %Q equivalent to a double quoted string, including “

HERE documents

```
document = << EOF
  some example text for the Ruby course.
  Several lines of “text” and #{variables}
EOF
```

- Multiline strings with ease
- Includes variable interpolation and ‘/” characters

Usual Suspects

- Common string operators and methods are supported
- Examples

```
'123' * 3        # => 123123123
'MiXed'.upcase!  # => MIXED
'MiXed'[0]       # => M
'a ' << 'test'   # => a test
```

- Ruby API has all the details

Symbols

- Common object used in Ruby
- References a constant string inside the Ruby symbol table
- Reduces complex String into a cheap integer operation
  - Fast lookup and comparison

```
hash = Hash.new
hash[:dog] = ‘fred’
hash[:cat] = ‘bonny’
```

Arrays

- Sequential position indexed data structure
- []

```
array = [1, 2, 3, 4]
```

- Array.new

```
array = Array.new
4.times { |i| array << i + 1 }
```

- %w()

```
array = %w( this is a new array )
# => [ ‘this’, ‘is’, ‘a’, ‘new’, ‘array’ ]
```

Usual Suspects

- Common array operations supported

```
a = [ 1, 2, 3, 4 ]
a << 5  # => [ 1, 2, 3, 4, 5 ]

a[-1]   # => 5
a[2, 3] # => [2, 3]

a.each_with_index do |o, i|
  puts "object at position #{i} is #{o}"
end
```

Usual Suspects

- Set operations

```
a = [ 1, 1, 2, 2, 3, 3, 4 ]
b = [ 5, 5, 4, 4, 3, 3, 2 ]

a | b # => [ 1, 2, 3, 4, 5 ] # unique values
a & b # => [ 2, 3, 4 ] # intersection
```

Hashes

- Unordered key/values pairs
- {}

```
hash = { :dog => ‘fred’, :cat => ‘bonny’ }
```

- Hash.new

```
hash = Hash.new
hash[:dog] = ‘fred’
hash[:cat] = ‘bonny’
```

- {} are optional, sometimes

```
pets.define(:dog => ‘fred’, :cat => ‘bonny’)
```

Usual Suspects

- Common hash operations supported

```
h = { :dog => 'fred', :cat => 'bonny' }
h.each do |k, v|
  puts "My #{k} name is #{v}"
end

h[:turtle] = 'preston'

h.has_key? :rabbit # => false
h.size             # => 3
h.values           # => [ 'fred', 'bonny', 'preston' ]
```

True, False and nil

- true, false and nil are all keywords in Ruby
- Everything is true, except for the keyword false and nil
- There is no Boolean class in Ruby, when required
  - nil behaves like false
  - any object except for nil and false behave as true

Examples of truth

```ruby
# hash returns nil if no key matches
name = pets[:dog]
puts "i have a dog, its name is #{name}" if name

# any object can be tested if its nil?
puts "no dogs unfortunately" if name.nil?

# string evaluating to a boolean
if a
  puts "the string 'false' evaluates to true"
end
```

Objects

- Everything we've covered so far has been an object
- No fundamental types like in C, C++, Java, etc
- All work with objects are done via references
- Arguments as always passed by reference, not value
  - Methods can modify objects outside their scope

References

```ruby
# creates a new string called Ruby, referenced by s
s = "Ruby"

# creates a new reference to s called t, same object
t = s

# modifies the object, affects both t and s
s << ' is cool'

# t now points to a new object
t = 'Rails'

# => Ruby is cool on Rails
puts "#{s} on #{t}"
```

Types and Reflection

- Each object has a type
  - Object is the root of all class hierarchies
    - class, superclass
- No casting required for references, dynamic typing
- Object: instance_of? is_a?, kind_of? and ===
- Class: ancestors

Examples

```
dog = Dog.new('Ben')

dog.class             # => Dog
dog.class.superclass  # => Object
dog.class.ancestors   # => [Dog, Object, Kernel]

Dog === dog           # => true
dog.is_a? Dog         # => true

dog.kind_of? Cat      # => false
```

Assignment

```
# Single

a = "cassie"

# Parallel

dog, cat = "Cassie", "Alf"
dog, cat = cat, dog

# Abbreviated

pets ||= %w( Dog Cat Rabbit )
```

Statements

conditionals and looping

the if and unless statements

```
if expression
  code
elsif expression # note the missing ‘e’
  code
else
  code
end


puts "We have some Ruby programmers!" if rubyists.size > 0

puts "We have some non-Ruby programmers!" unless
rubyists.size > 0
```

the case statement

```
case object
when String
when Numeric
when TrueClass, FalseClass
else "other"
end
```

Both case and if return a value
to its caller

the for loop

```
array = %w( an array of strings )

for word in array do
  puts word
end

# Also equivalent

array.each do |word|
  puts word
end
```

Blocks

Smalltalk'isms in Ruby

Blocks

- Fundamental to Ruby's style and conventions
- Segment of code, that can be passed as an argument to any method call
- Identified by a do...end or { .. } at the end of a method
- Can accept variables, and return a value
- Accessible by the developer using yield or block.call

Examples

```ruby
1.upto(10) do |i|
  puts i
end
```

```ruby
cars.each do |vehicle|
  vehicle.send :register
end

File.open('book.txt','w') do |f|
  f << content
end
```

Block Development

```
class Car
  def parts
    @parts.each do |part|
      yield part if block_given?
    end
  end
end

ferrari.parts do |part|
  puts "My ferrari has #{part}"
end
```

Closures

- Blocks are also closures
- Can reference variables outside the scope of the block, even after the those variables original scope has disappeared

Example

```
class Car
  def parts
    @parts.each do |part|
      yield part if block_given?
    end
  end

  parts = SpareParts.new

  ferrari.parts do |part|
    puts "My ferrari has #{part}"
    parts.ensure_available! part
  end
```

Enumerable

- select
  - Find all members where the block evaluates to true
- reject
  - Find all members where the block evaluates to false
- collect
  - Invoke the block on all members, return results array
- inject
  - Inject an object into a block called on all members

Select

```
cassie = Dog.new('Cassie')
alf    = Cat.new('Alf')
ben    = Dog.new('Ben')

pets   = [ cassie, alf, ben ]

pets.select { |pet| pet.is_a? Dog }

# => [ cassie, ben ]
```

Reject

```
cassie = Dog.new('Cassie')
alf    = Cat.new('Alf')
ben    = Dog.new('Ben')

pets   = [ cassie, alf, ben ]
```

```
pets.reject { |pet| pet.is_a? Cat }
```

```
# => [ cassie, ben ]
```

Collect

```
cassie = Dog.new('Cassie')
alf    = Cat.new('Alf')
ben    = Dog.new('Ben')

pets   = [ cassie, alf, ben ]
```

```
pets.collect { |pet| pet.breed }

# => [‘cocker spaniel’, ‘house cat’,
‘spaniel’]
```

Inject

Inject

```
cassie = Dog.new('Cassie')
alf    = Cat.new('Alf')
ben    = Dog.new('Ben')

pets   = [ cassie, alf, ben ]
```

```
pets.inject({}) {|m, pet| m[pet.breed] = pet; m}

# => { 'cocker spaniel' => cassie,
       'house cat' => alf,
       'spaniel' => ben }
```

Equivalent Iterator Style

```
import java.util.Map;
import java.util.HashMap;
import java.util.Iterator;

Map results = new HashMap();

for (Iterator i = pets.iterator(); i.hasNext(); ) {
  Pet p = (Pet) i.next();
  results.put(p.breed(), p);
}
```

Methods

Definitions, Visibility, Constructors, etc

Methods

- Defined with the keyword def, marked by the word end
- Methods return the value of the last statement
  - Unless terminated elsewhere via keyword return or an exception via raise
    - May return multiple values for parallel assignment
- self is the default receiver for methods
- Object#send is the same as a method invocation

Example

```
def purchase(amount, method)
  raise 'No amount specified' unless amount or amount < 0

  case method
  when CreditCard
    online_purchase(amount)
  when Cash
    cash_purchase(amount)
  else
    raise 'Unknown payment method'
  end
end
```

What's the last line in this method? ie the return value?

Method Conventions

- Method names are lowercase
- Method names use underscore notation
  - define_method, not defineMethod
- Method names may end in ? or !
  - ? indicates a boolean return type
  - ! indicates an direct modification of the receiver
- Parenthesis are optional for declaration and invocation

Default Values

```ruby
def purchase!(amount, method)
  # ...
end

car.purchase! 100_000, amex_card

def purchase!(amount, method = Cash)
  # ...
end

car.purchase! 100_000

def purchase!(amount = 0, method = Cash)
  # ...
end

car.purchase!
```

```
public class Car {

  public static final String DEFAULT_MAKE  = "Audi";
  public static final String DEFAULT_MODEL = "TTS";

  public Car() {
    this(DEFAULT_MODEL);
  }

  public Car(String make) {
    this(make, DEFAULT_MODEL);
  }

  public Car(String make, String model) {
    this(make, model, new Date());
  }

  public Car(String make, String model, Date year) {
    this.make = make;
    this.model = model;
    this.year = year;
  }

  private String make;
  private String model;
  private Date year;
}
```

Java Chaining

Ruby

```ruby
class Car
  attr_accessor :make, :model, :year

  def initialize(make = 'Audi', model = 'TTS', year = Time.now)
    @make  = make
    @model = model
    @year  = year
  end
end
```

Named arguments

```ruby
def purchase(options = {})
  amount = options[:amount]
  method = options[:method]
  # ...
end
```

```ruby
car.purchase :amount => 100_000, :method => Card
```

- No direct support for named arguments in Ruby
- Hash braces are optional { }, along with parenthesis

Classes

Definitions, Visibility, Constructors, etc

Classes

- Defined with the keyword class, marked by end
- Single inheritance only via the < construct
  - Multiple module inclusion via include statement
  - Object is the default and root superclass
- Constructor always called initialize, 0..n arguments
- Methods public by default

Classes & Methods

- Operators can be overloaded, eg. +, [], /, *, etc
- Existing classes can be extended with new methods
- Dynamic method table
  - Allows catching and responding to missing methods
- Duck typing, no casting or type matching required
  - If it responds to a method, thats acceptable

Dynamically adding methods

```
class String
  def parenthesize
    "(#{self})"
  end
end

"Ruby!".parenthesize # => “(Ruby!)”

# or

1.day.ago
5.days.from_now
```

Variables

- Instance variables start with a @variable_name
- Class variables start with @@variable_name
- Getters/Setters can be dynamically defined
  - attr_accessor, attr_reader and attr_writer
- Instance variables are always private
- Constants are always public

Duck Typing #1

```
class Dog
  attr_accessor :name, :breed
  def initialize(name, breed)
    @name, @breed = name, breed
  end
end

class Cat
  attr_accessor :name, :breed
  def initialize(name, breed)
    @name, @breed = name, breed
  end
end
```

Duck Typing #2

```
class Pets
  attr_accessor :pets

  def initialize(*pets)
    @pets = pets
  end

  def to_s
    pets = @pets.collect do |pet|
      "#{pet.breed}: #{pet.name}"
    end
    pets.join(', ')
  end
end
```

Duck Typing #3

```
cassie = Dog.new('Cassie', 'Cocker Spaniel')
alf    = Cat.new('Alf', 'House Cat')
ben    = Dog.new('Ben', 'Spaniel')

pets   = Pets.new cassie, alf, ben

puts pets # => Cocker Spaniel: Cassie, House Cat:
Alf, Spaniel: Ben
```

Ruby Koans

Writing some code!

Twitter 1.0

Command Line Version

- Social networking micro-blogging platform
  - 140 character limit on posts
- Thousands of contributors around the world
- Fastest networking platform available

Interface

- Command Line Client
- Twitter Gem
  - Abstracts net/http networking access to Twitter
    - README is in /Library/Ruby/Gems/1.8/gems/twitter-0.4.2 describing the API
- Access the public timeline
  - Display the 50 most recent public tweets

Design

- Require rubygems and the twitter gem
- Create a Twitter class, with a method public_tweets
- Use the Twitter gem to access the public timeline
- Display the tweets to the console

Example

```
require 'rubygems'
require 'twitter'

class Tweeter

  def public_tweets
    fetch_tweets.each do |s|
      puts "#{s.user.name}: #{s.text}"
    end
  end

  private

    def fetch_tweets
      Twitter::Base.new('', '').timeline(:public)
    end

end

Tweeter.new.public_tweets
```

Further enhancements

- Support for reading a particular users tweets
- Support for posting new tweets from the command line
  - Requires valid credentials/account