Elegant Code with Ruby

By: Ben Hughes

me@benhughes.name

Elegance Matters

- Enjoyment
- Maintainability
- Agility
- Developer Expressivity
- Intrinsic Beauty of Well-Engineered Code

Introducing Ruby

"... the subtle elegance of the Ruby idiom is a slowly appreciated and highly satisfying flavor."
- Edd Dumbil

So why does Ruby encourage elegance?

1. **Very Dynamic**
2. **Rich & Flexible Syntax**

Ruby Dynamism

- Very Object-Oriented
- Extensive Support for Code Blocks/Closures
- Open Classes
- Powerful Meta Programming
- Dynamic Code Evaluation
- Executable Declarations

Rich & Flexible Syntax

- Not Strict
- Parenthesis Optional
- Concise Syntax for Code Blocks

Code Blocks / Closures

- Very Pervasive in the Ruby Idiom
- "Closures" describe the variable scoping implicit in the way code blocks work.
- Aka. Lambdas or Anonymous Functions
- Arcane or advanced? JavaScript supports closures!
- Helps support a **functional** style of programming.

```
do |arg1, arg1|

end
```

```
{|arg1, arg2|   }
```

Code Block Examples

```
people.each do |person|
  puts person.name
end
```

```
1.upto(100) do |i|
  puts i
end
```

```
names = people.map {|p| p.name }
```

```
age_sum = people.inject(0) {|sum, p| sum += p.age }
```

```
drinkers = people.select do |person|
  person.age >= 21 or person.doesnt_care?
end
```

```
open 'entries.txt' do |file|
  file.each_line do |line|
    process_entry(line)
  end
end
```

Classes are Open

```
class NilClass
  def blank?
    false
  end
end
```

```
class String
  def blank?
    self == ''
  end
end
```

```
# Using some methods added core classes by Rails...

10.days_ago
2.minutes + 30.seconds
Time.now + 1.day + 13.hours + 4.minutes + 9.seconds
Time.now.last_month
```

```
class Fixnum
  def +(another)
    self - another
  end
end

# 4 + 2
# => 2
```

Add Methods to Individual Objects

```
ben = Person.new('Ben')

def ben.send_message(message)
  Mailer.send_message(self.name, message)
end
```

```
image = Image.get('my_image')
image.extend(JpegCapabilities) if image.type == 'jpeg'
```

Executable Declarations

Classes can assist in building
and declaring themselves...

```
class Person
  attr_accessor :name
end
```

```
class Order < Base
  belongs_to :customer
  has_many :order_items
end
```

```
class Person
  def name
    @name
  end

  if ENV['ENABLE_SUPERPOWERS']
    def give_superpowers
      # Method body here...
    end
  end
end
```

```
class Category < Base
  acts_as_tree
end
```

Domain-Specific Languages (DSLs)

- "Sub-Languages" of Ruby for specific purposes.
- Reduces complexity of the system exposed.
- Captures software knowledge in a very readable form.
- Ruby is particularly well-suited for DSL building:
  - Loose but concise syntax
  - Powerful meta programming capabilities

Some Examples of Ruby Elegance

- Markaby - Markup as Ruby
- ActiveRecord - Object-Relational Mapping
- RSpec - Elegant Behavior Specification
- JazzToolbox - Music Relationships
- Ambition - Enumerable Proxy API

Markaby - Markup as Ruby

```ruby
mab.html do
  head { title "Boats.com" }
  body do
    h1 "Boats.com has great deals"
    ul do
      li "$49 for a canoe"
      li "$39 for a raft"
      li "$29 for a huge boat"
    end
  end
end
```

```html
<html>
  <head><title>Boats.com</title></head>
  <body>
    <h1>Boats.com has great deals</h1>
    <ul>
      <li>$49 for a canoe</li>
      <li>$39 for a raft</li>
      <li>$29 for a huge boat</li>
    </ul>
  </body>
</html>
```

```ruby
div.page! do
  div.content! do
    h1 "A Short Short Saintly Dog"

    p.entry "A test entry title..."
    p.entry "Another entry..."
  end
end
```

```html
<div id="page">
  <div id="content">
    <h1>A Short Short Saintly Dog</h1>
    <p class="entry">A test entry title...</p>
    <p class="entry">Another entry...</p>
  </div>
</div>
```

ActiveRecord (ORM)

```
class Manager < ActiveRecord::Base
  has_many :projects
  has_many :deliverables, :through => :projects

  validates_presence_of :name
end

class Project < ActiveRecord::Base
  belongs_to :manager
  belongs_to :category
  has_many :deliverables

  validates_presence_of :name
  validates_associated :deliverables

  before_create :send_notification

  def send_notification
    NotificationMailer.send_new_project_notification(self)
  end
end

class Deliverable < ActiveRecord::Base
  belong_to :project
  named_scope :due_after, lambda {|date| {:conditions => ["due_date > ?", date] } }
  default_scope :order => :due_date
end
```

```ruby
Project.count  # => SELECT COUNT(*) FROM projects

@project = Project.first
@project.deliverables.last
# => SELECT * FROM deliverables WHERE project_id = 1 ORDER BY due_date DESC LIMIT 1

Manager.create(:name => 'Ben Hughes')
# => INSERT INTO managers (name) VALUES ('Ben Hughes')

Project.where(:id => [1, 2, 3]).with(:manager).order(:name)
# SELECT projects.*, managers.* FROM projects
# INNER JOIN managers ON projects.manager_id = managers.id
# WHERE id IN (1, 2, 3) ORDER BY name

@manager = Manager.find_by_name("Ben Hughes")
@manager.deliverables.due_after(3.days.from_now)

# SELECT deliverables.* FROM deliverables
# INNER JOIN projects ON deliverables.project_id = projects.id
# INNER JOIN managers ON projects.manager_id = managers.id
# WHERE managers.name = 'Ben Hughes' AND delivearbles.due_date > '2009-04-18'
# ORDER BY due_date

project = Manager.find_by_name("Chris Horn").projects.create(
  :name => 'Zombie Apocalypse',
  :deliverables_attributes => [
    { :name => "Short-Term Survival", :due_date => 5.hours.from_now },
    { :name => "Mid-Term Survival", :due_date => 10.days.from_now },
    { :name => "Long-Term Survival", :due_date => 6.years.from_now }
  ]
)

# INSERT INTO projects (name, manager_id) VALUES ('Zombie Apocalypse', 1)
# INSERT INTO deliverables (project_id, name) VALUES (1, 'Short-Term Survival')
# ...
# [Email is sent through NotificationMailer]
```

Behavior Specification with RSpec

```ruby
describe Bowling do
  it "should score 0 for gutter game" do
    bowling = Bowling.new
    20.times { bowling.hit(0) }
    bowling.score.should == 0
  end
end
```

```ruby
describe PostsController do
  context "when logged in" do
    before { login_as(:ben_hughes) }

    describe "listing posts" do
      before { get :index }
      it { should respond_with(:success) }
      it { should render_template(:index) }
      it { should assign_to(:posts) }
    end

    describe "creating a post" do
      it "should change total count" do
        lambda do
          post :create, :post => {
            :title => 'Test',
            :author => 'Ben Hughes',
            :body => 'Lorem'
          }
        end.should change('Post.count').by(1)
      end
    end
  end
end
```

Music Relationships with JazzToolbox

```
Chord['Ebmaj7'].notes
Chord['maj7'].in_key_of('Eb').notes
# => ['Eb', 'G', 'Bb', 'D']
```

```
Scale['Whole Tone'].notes
# => ['C', 'D', 'E', 'F#', 'G#', 'Bb']
```

```
Scale['Major'].in_key_of('Eb').modes['Dorian'].notes
# => ['F', 'G', 'Ab', 'Bb', 'C', 'D', 'Eb']
```

```
Scale['Major']['Dorian'].chords.symbols
# => ['min7', 'min6']
```

```
Chord['Amin7'].modes.names
# => ['A Dorian']
```

Ambition - Enumerable Proxy API

```
User.select {|m| (m.name == 'jon' || m.name == 'rick') && m.age == 21 }

# => SELECT * FROM users WHERE
#    ((users.name = 'jon' OR users.name = 'rick') AND users.age = 21)
```

```
User.select {|m| [1, 2, 3, 4].include? m.id }

# => SELECT * FROM users WHERE users.id IN (1, 2, 3, 4)
```

```
User.select {|m| m.name == 'jon' }.sort_by {|m| -m.age }.first(5)

# => SELECT * FROM users WHERE users.name = 'jon' ORDER BY users.age DESC LIMIT 5
```

```
User.select do |u|
  u.email == 'me@benhughes.name' && u.profile.name == 'Ben Hughes'
end

# => SELECT * FROM users
#    LEFT OUTER JOIN profiles ON profiles.user_id = users.id
#    WHERE ((users.email = 'me@benhughes.name' AND profiles.name = 'ben hughes'))
```

No Perfect Language

- Everything involves tradeoffs - Ruby and other languages have benefits as well as costs.
- Other dynamic languages exist that embody many of the concepts in this presentation.

Conclusion

- Elegance Matters
- Expanded Paradigm for Problem Solving
- Learning Ruby in-depth is a positive experience
- “Power with clarity and choice.” - Glenn Vanderburg

Thats it!
Questions?

By: Ben Hughes
RubyElegance.com

ben@railsgarden.com