Advanced Ruby Idioms So Clean You Can Eat Off Of Them

Brian Guthrie
ThoughtWorks®
bguthrie@thoughtworks.com
http://twitter.com/bguthrie

handshake

resource_full

awsymandias

What this talk is about

“magic”

http://www.flickr.com/photos/cayusa/2962437091/

encouraging you to try things

preventing you from poking yourself with the sharp end of a pointed stick

magic

magic

technical understanding

design maturity

What this talk is not about

trying to stop you

explaining why

Who this talk is for

Who this talk is for

0-6 months

6-12 months

12-36 months

36+ months

Who this talk is for

Who this talk is for

0-6 months

6-12 months

12-36 months

36+ months

total
noob sauce

hardened
ruby veteran

Who this talk is for

0-6 months

6-12 months

12-36 months

36+ months

total noob sauce

a danger to yourself and others

hardened ruby veteran

Who this talk is for

0-6 months

6-12 months

12-36 months

36+ months

total noob sauce

a danger to yourself and others

hardened ruby veteran

You know who you are.

Who this talk is for

0-6 months

6-12 months

12-36 months

36+ months

total noob sauce

total lack of curiousity

thick controllers (total noob sauce)

ask questions anytime

http://www.flickr.com/photos/jonnya/4212907371/

task:
build a custom validation framework

task:

build a custom validation framework

a danger to yourself
and others

```ruby
class Pirate < BrianRecord::Base
  validates_presence_of :parrot
  attr_reader :parrot
  def initialize(parrot=nil)
    @parrot = parrot
  end
end
```

```ruby
it "bangs if not valid" do
  lambda do
    Pirate.new.valid?
  end.should raise_error("Bang!")
end
```

```ruby
class BrianRecord::Base
  class << self
    def validators
      @validators ||= []
    end

    def validates_presence_of(attribute, opts={})
      validators << lambda do |object|
        return !object.send(attribute).nil?
      end
    end

    def valid?
      self.class.validators.each do |validator|
        is_valid = validator.call(self)
        raise "Bang!" unless is_valid
      end
    end
  end
```

```ruby
def validates_presence_of(attribute, opts={})
  validators << lambda do |object|
    return !object.send(attribute).nil?
  end
end
```

```ruby
def validates_presence_of(attribute, opts={})
  validators << lambda do |object|
    if opts[:allow_blank]
      ...
    end

    if opts[:if] || opts[:unless]
      ...
    end

    if attribute.to_s =~ /_id/
      ...
    end

    return !object.send(attribute).nil?
  end
end
```

```ruby
def validates_presence_of(attributes, opts={})
  attributes.each do |attr|
    validators << lambda do |object|
      ...
      return !object.send(attribute).nil?
    end
  end
end
```

```ruby
def validates_presence_of(attributes, opts={})
  validators << lambda do |object|
    return !object.send(attribute).nil?, "expected #{attribute} to not be nil"
  end
end
```

```ruby
class BrianRecord::Base
  class << self
    def validators
      @validators ||= []
    end

    def validates_presence_of(attribute, opts={})
      validators << BrianRecord::PresenceOfValidator.new(attribute, opts)
    end
  end

  def valid?
    self.class.validators.each do |validator|
      is_valid = validator.validate(self)
      raise validator.message unless is_valid
    end
  end
end
```

```ruby
class BrianRecord::PresenceOfValidator
  def initialize(attribute, opts={})
    @attribute = attribute
  end

  def message
    "expected #{@attribute} to not be nil"
  end

  def validate(object)
    return !object.send(@attribute).nil?
  end
end
```

Tip

Offload the heavy stuff to dedicated objects.

It's worth the trouble.

```ruby
class Pirate < BrianRecord::Base
  validates_presence_of :parrot
  attr_reader :parrot
  def initialize(parrot)
    @parrot = parrot
  end
end
```

```ruby
class Captain < Pirate
  validates_presence_of :peg_leg
  attr_reader :peg_leg
  def initialize(parrot, peg_leg)
    @parrot, @peg_leg = parrot, peg_leg
  end
end
```

```ruby
Captain.new(nil, "wooden").valid? #=> bang?
```

```ruby
class BrianRecord::Base
  class << self
    def validators
      @validators ||= []
    end
  end
end
```

```ruby
class BrianRecord::Base
  class << self
    def validators
      @validators ||= if superclass.respond_to?(:validators)
        superclass.validators.dup
      else [] end
    end
  end
end
```

```
class_inheritable_accessor
class_inheritable_array
class_inheritable_hash
```

Tip

Make sure your subclasses honor the behavior of their superclasses.

It's worth the trouble.

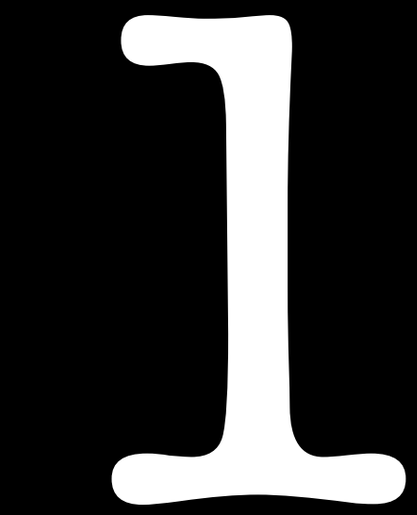
a brief detour

http://www.flickr.com/photos/garyseven/4434835473/

6

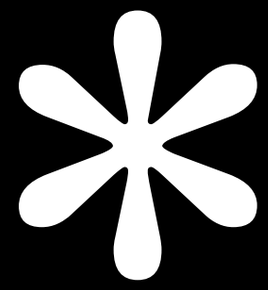
(acceptable)
types of
Ruby
method signatures

0



{}

1 {}



012H1H*

0 1 2 ... n

```
def this(method, signature, is, too, brittle)
```

H 1H *

```ruby
class User
  def initialize(attributes={})
    @username, @password = attributes.values_at(:username, :password)
  end
end
```

```ruby
class ActiveRecord::Base
  class << self
    def validates_presence_of(attribute, opts={})
      opts.reverse_merge!(:some_default => "value")
      validators << PresenceOfValidator.new(attribute, opts)
    end
  end
end
```

```ruby
class ActiveRecord::Base
  class << self
    def validates_presence_of(*attributes)
      opts = attributes.extract_options!
      attributes.each do |attribute|
        validators << PresenceOfValidator.new(attribute, opts)
      end
    end
  end
end
```

Tip

Stick to the Six Acceptable Ruby Method Signatures.

0, 1, 2, H, 1H, *

```ruby
it "bangs if it isn't valid for the particular attribute we're interested in" do
  lambda do
    Pirate.new(nil).valid_for_parrot? #=> bang?
  end.should raise_error("expected parrot not to be nil")
end
```

```
Pirate.new(nil).valid_for?(:parrot)
```

a danger to yourself
and others

```ruby
def method_missing(method_name, *args, &block)
  if (match = method_name.to_s.match(/valid_for_(.*)\?/))
    attribute = match[1]
    self.class.validators.select do |v|
      v.attribute.to_s == attribute.to_s
    end.each do |validator|
      raise validator.message unless validator.validate(self)
    end
  else super end
end

def respond_to?(method_name)
  method_name.to_s.match(/valid_for_(.*)\?/) || super
end
```

```ruby
def validators
  @validators ||= if superclass.respond_to?(:validators)
    superclass.validators.dup
  else [] end
end

def validators_for(attribute)
  validators.select do |v|
    v.attribute.to_s == attribute.to_s
  end
end

def validates_presence_of(attribute, opts={})
  add_validator(attribute,
    BrianRecord::PresenceOfValidator.new(attribute, opts))
end

def add_validator(attribute, validator)
  validators << validator
  self.instance_eval do
    define_method "valid_for_#{attribute}?" do
      self.class.validators_for(attribute).each do |validator|
        raise validator.message unless validator.validate(self)
      end
    end
  end
end
```

```ruby
def add_validator(attribute, validator)
  validators << validator
  # def valid_for_parrot?
  #   self.class.validators_for(:parrot).each do |validator|
  #     raise validator.message unless validator.validate(self)
  #   end
  # end
  self.class_eval <<-RUBY
    def valid_for_#{attribute}?
      self.class.validators_for(:#{attribute}).each do |validator|
        raise validator.message unless validator.validate(self)
      end
    end
  RUBY
end
```

Tip

Add behavior with
instance_eval or define_method,
not method_missing.

http://www.flickr.com/photos/suneko/389852016/

+

http://www.flickr.com/photos/contusion/4066189755/

validations anywhere!

```ruby
class Object
  class << self
    def validators
      @validators ||= if superclass.respond_to?(:validators)
        superclass.validators.dup
      else [] end
    end

    def validators_for(attribute)
      validators.select do |v|
        v.attribute.to_s == attribute.to_s
      end
    end

    ...
  end
end
```

```
>> Object.new.method :valid?
=> #<Method: Object#valid?>
```

```ruby
module BrianRecord
  module Extensions
    module Validations
      module ClassMethods
        def validators
          @validators ||= if superclass.respond_to?(:validators)
            superclass.validators.dup
          else [] end
        end

        def validators_for(attribute)
          validators.select do |v|
            v.attribute.to_s == attribute.to_s
          end
        end

        ...
      end

      module InstancMethods
        def valid?
          ...
        end
      end
    end
  end
end
```

```ruby
module BrianRecord
  module Extensions
    module Validations
      def self.included(base)
        base.send :include, InstanceMethods
        base.send :extend, ClassMethods
      end
    end
  end
end
```

```
>> Object.new.method :valid?
=> #<Method: Object(BrianRecord::Extensions::Validations::InstanceMethods)#valid?>
```

Tip

Modularize and name any extensions you build.

Some poor sucker, like me, has to maintain that thing.

Tip

Use self.included to leverage existing class behaviors.

Purely my own opinion.

```ruby
module Extensions
  module Object
    module Tap
      def tap
        raise unless $proprietary_global
        yield self if block_given?
        self
      end
    end
  end
end
```

a_very_bad_gem/lib/extensions/object/tap.rb

```ruby
unless defined?(tap)
  def tap
    yield self if block_given?
    self
  end
end
```

Tip

Extend respectfully:
Check for competing implementations.

Or I will hunt you down and slay you.

Tip

Write tests.

Write tests.

Write tests.

Write tests.

Write tests.

Write tests.

Write tests.

Write tests.

Write tests.

Write tests.

Write tests.

YUM

RUBY WILL EAT YOU

YUM

http://www.flickr.com/photos/pandiyan/146700348/

YUM

http://www.flickr.com/photos/bunchofpants/52816417/

```
public class StringUtil {
        public static boolean isBlank(Object o) {
                ...
        }
}

public class XmlUtil {
        public static NodeList search(Node node, String xpath) {
                ...
        }
}

public class DateUtil {
        public static Date yesterday() {
                ...
        }
}
```

Brian Guthrie
ThoughtWorks®
bguthrie@thoughtworks.com
http://twitter.com/bguthrie

questions?