Meta-Programming in Ruby

Brian Sam-Bodden

@bsbodden

Ruby

Ruby Ideals...

Programming should be fun!

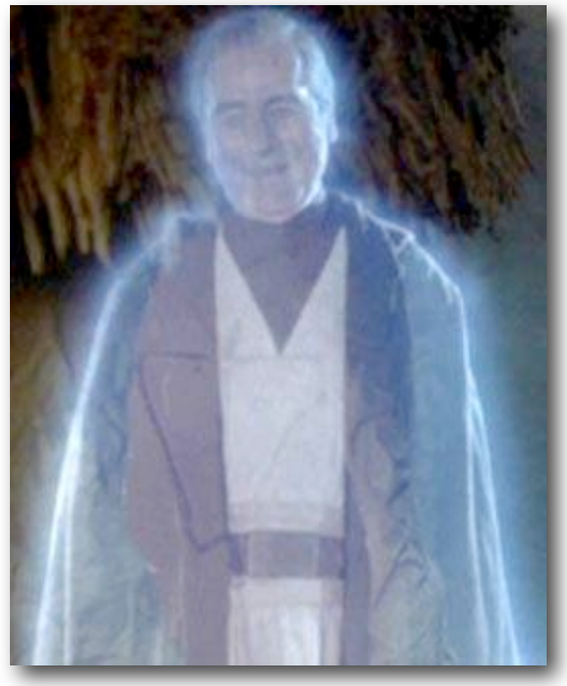
Java Developer :-)

A programming language should treat you like a responsible adult!*

Meta-Programing

Too little

Too much



Just Enough

Without **Ruby** there would be no Rails

Ruby's meta-programing is the key feature that makes Rails magical

Open Classes

Say we have an Array like:

[1,2,3,4,5]

and we want to sum of
all of its elements

Let's try something like:

[1,2,3,4,5].sum

Terminal — ruby — 64x6

```
brian-sam-boddens-computer:~ bsbodden$ irb
irb(main):001:0> [1,2,3,4,5].sum
NoMethodError: undefined method `sum' for [1, 2, 3, 4, 5]:Array
        from (irb):1
irb(main):002:0>
```

Rats! Doesn't work in **Ruby**

… yet!

Ruby classes are 

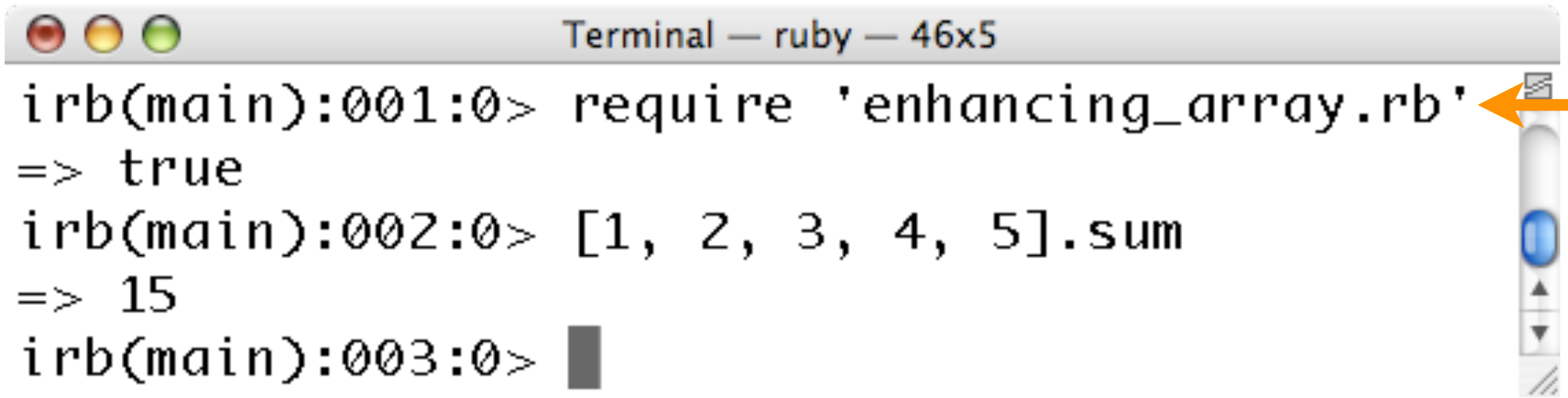
We can teach the Array class a new behavior

enhancing_array.rb

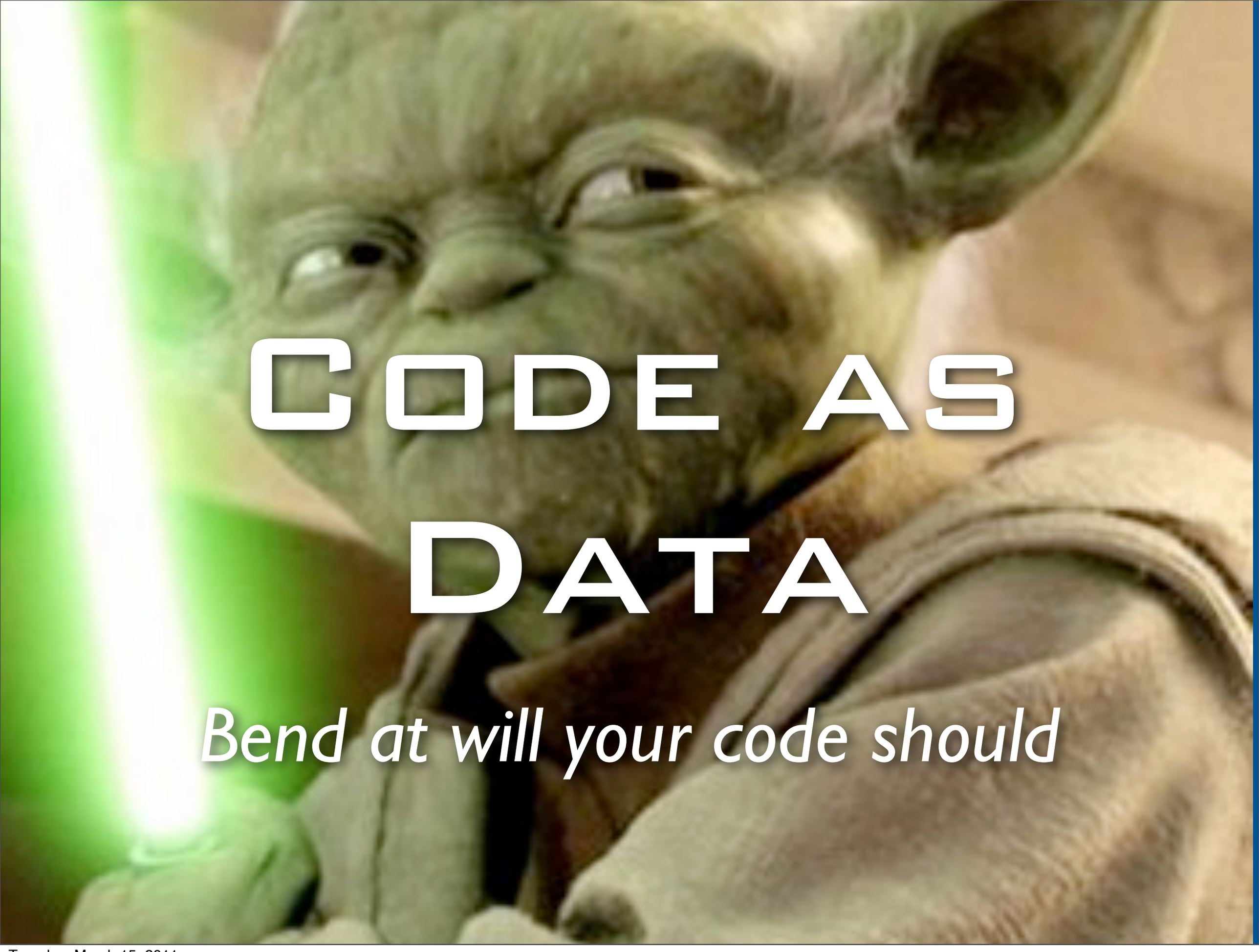
```ruby
class Array
  def sum
    inject(0) { |s, v| s + v }
  end
end
```

Line: 5 Column: 4 Ruby Soft Tabs: 2

Let's try again!

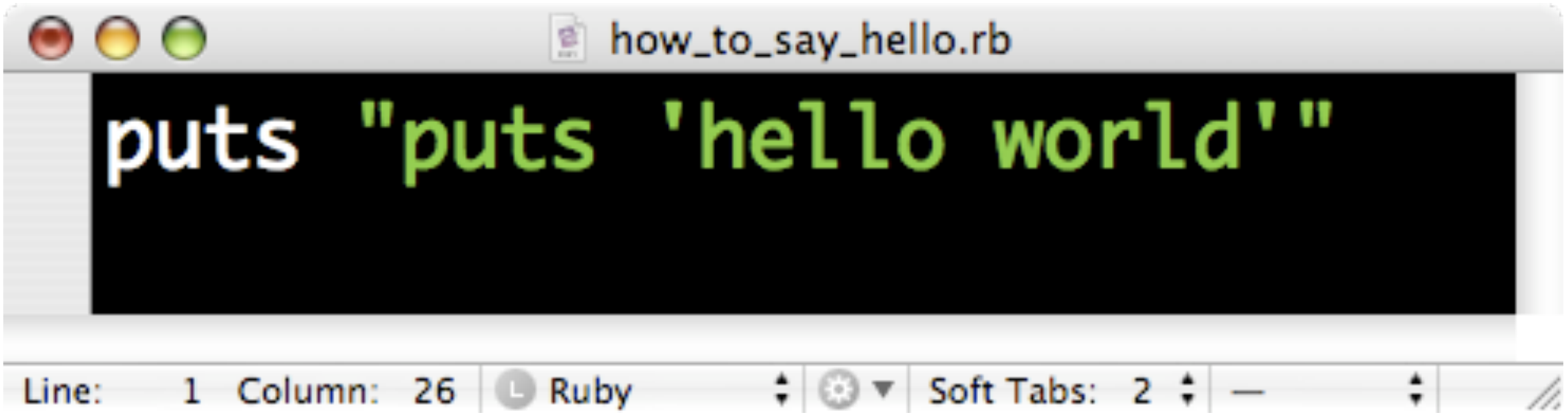


All you need to do is load the file containing the enhancements

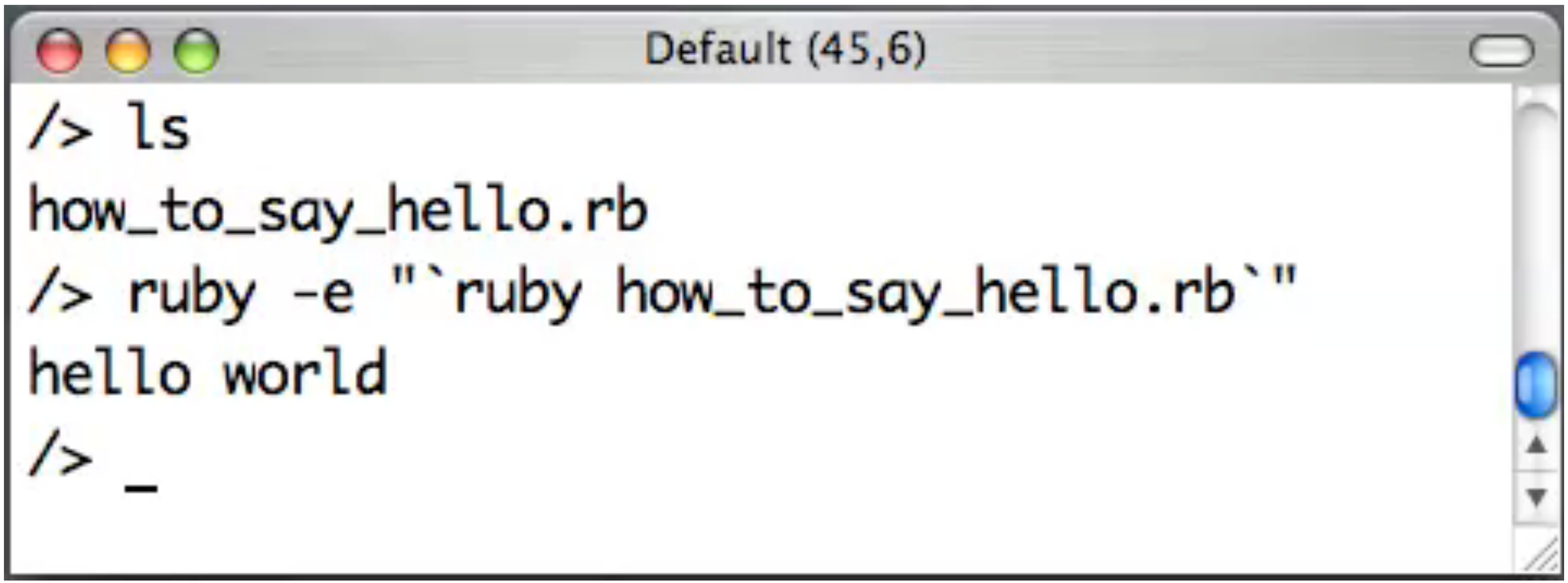

Code as
Data
Bend at will your code should

When code can be manipulated as data
a whole world of possibilities opens

In **Ruby** you can evaluate the
contents of a string

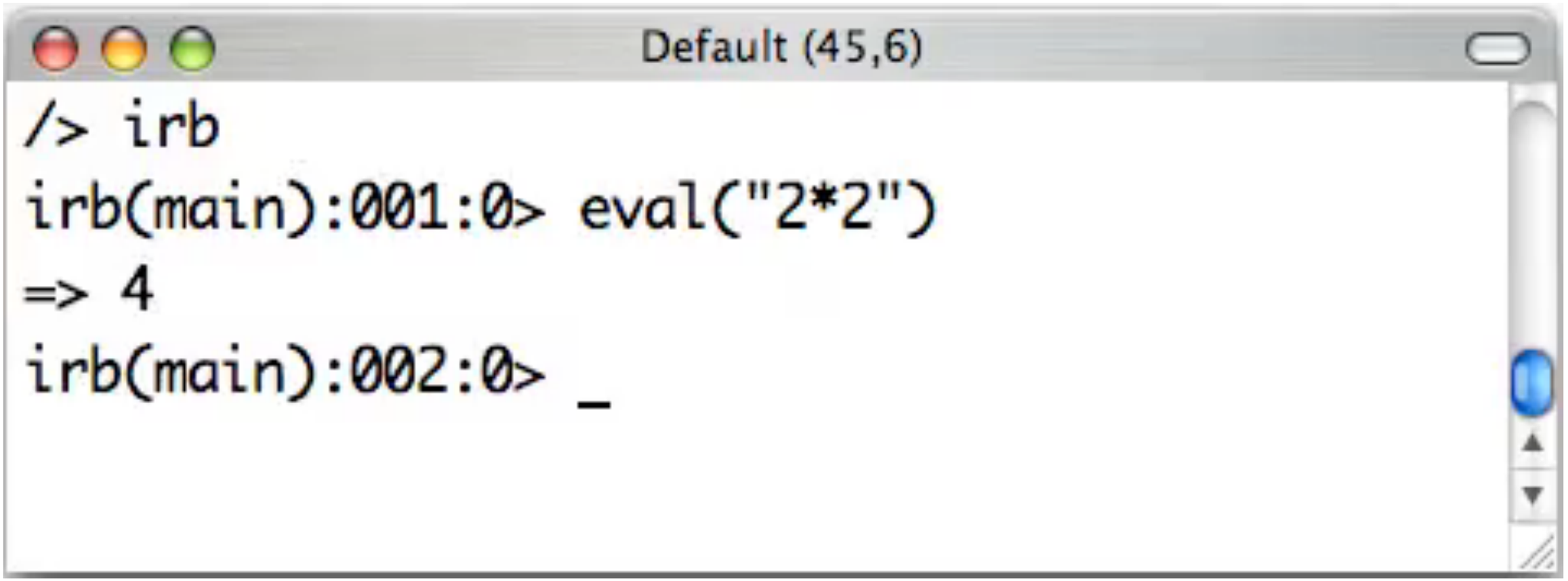


Code that invokes code, that invokes code



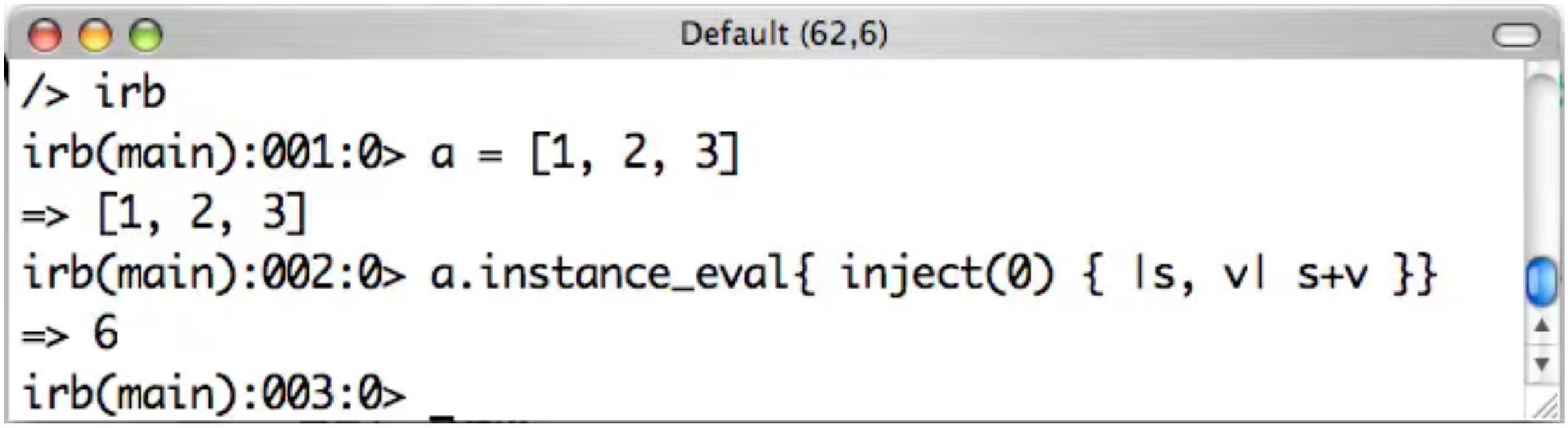
There you go, there you go

Ruby provides the eval family of methods for runtime execution of code stored in strings



eval will evaluate any string

instance_eval can evaluate a string or a code block



in the context of the receiver

class_eval.rb

```ruby
clazz = Class.new
clazz.class_eval do
  def wheres_the_beef?
    puts "in here"
  end
end

instance_of_clazz = clazz.new
instance_of_clazz.wheres_the_beef?
```

Line: 9 Column: 18 Ruby Soft Tabs: 2 whe...

class_eval can evaluate a string or a code block in the context of the class or module it is called on

Default (22,3)

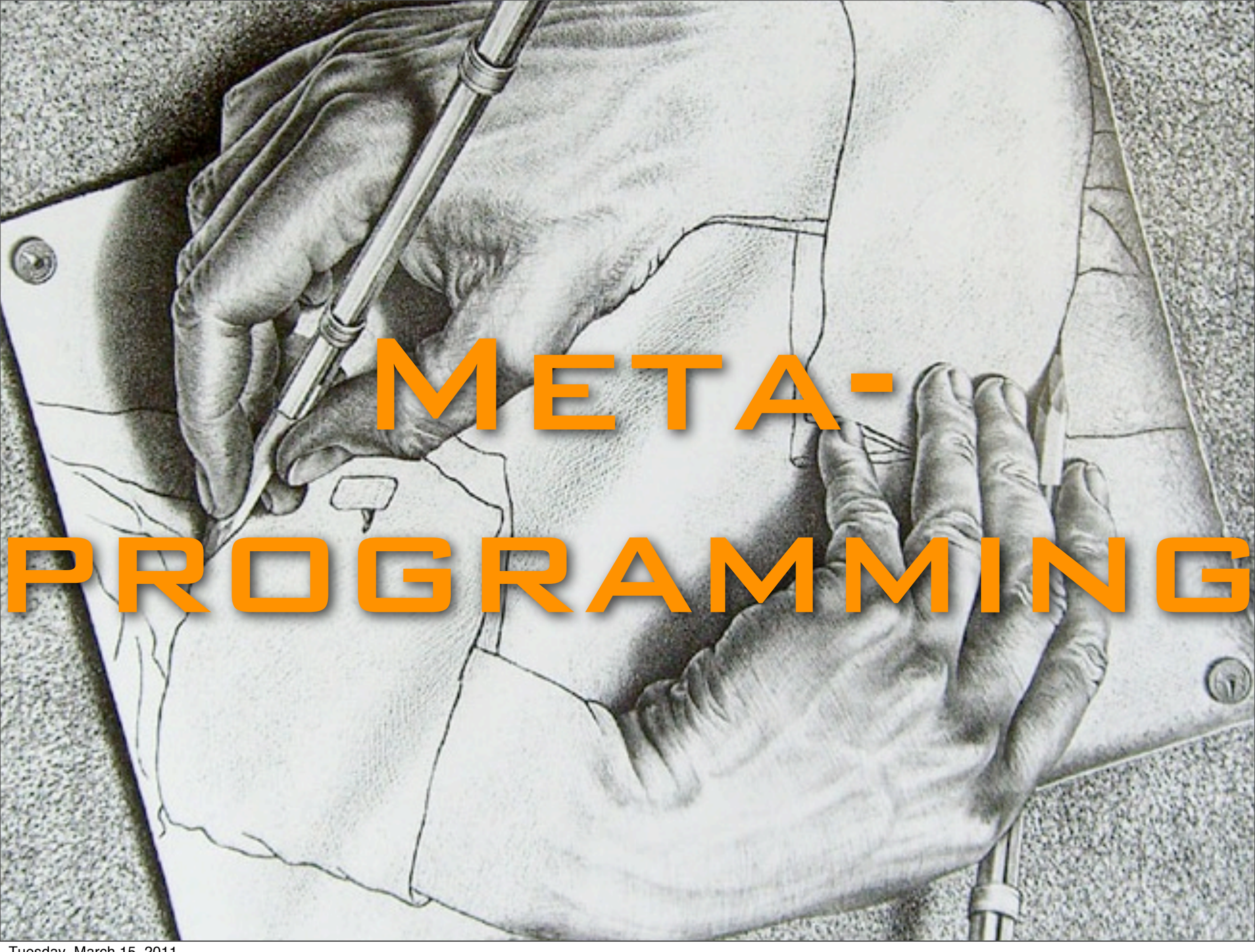
```
/> ruby class_eval.rb
in here
/> _
```

In **Ruby** we try to think about sending a message to an object rather than calling a method on an object

Default (59,11)

```
/> irb
irb(main):001:0> my_string = "yo!"
=> "yo!"
irb(main):002:0> my_other_string = String.new("yo!")
=> "yo!"
irb(main):003:0> yet_another_one = String.send(:new, "yo!")
=> "yo!"
irb(main):004:0> _
```

In simple cases (above) syntactic sugar hides it, but we can use it in interesting ways...


META-
PROGRAMMING

Meta-programming

...is about programs that write programs

...it's a superb tool for building frameworks

...it's the key ingredient for building domain-specific languages

Ruby's is a great vehicle for meta-programming because it is:

dynamic and **r**eflexive

open and **m**alleable

code is **d**ata, **d**ata is **c**ode

clean* **s**yntax

programming **e**vent **m**odel

Ruby on Rails

...uses meta-programming to bring the language closer to the problem

...is a collection of domain-specific languages (DSL) for building web applications

Singleton Class

In **Ruby** you can enhance a particular **instance** of a class

Ruby uses a **proxy** class known as the **singleton** class

Meta-class: The **singleton** for **Class** objects

class_eval indirectly uses the meta-class

person.rb

```ruby
class Person

  def self.make_greeting(language, greeting)
    class_eval "def greet_in_#{language}; puts '#{greeting}'; end"
  end

  make_greeting("spanish", "Hola!")
  make_greeting("english", "Hello!")
end
```

Line: 11 Column: 1 Ruby Soft Tabs: 2 self.make_greeting(language, greeting)

adding a singleton method to the class

Rails relies heavily on **Ruby's** ability to dynamically enhance a class

Default (79,16)

```
irb(main):003:0> bob.greet_in_spanish
Hola!
=> nil
irb(main):004:0> bob.greet_in_english
Hello!
=> nil
irb(main):005:0> clazz = Person
=> Person
irb(main):006:0> clazz.make_greeting('dutch', 'Hallo!')
=> nil
irb(main):007:0> jim = clazz.new
=> #<Person:0x57214>
irb(main):008:0> jim.greet_in_dutch
Hallo!
=> nil
irb(main):009:0> _
```

Shortcuts such as *define_method* exist to make life easier

person2.rb

```ruby
class Person

  def self.make_greeting(language, greeting)
    define_method "greet_in_#{language}" do
      puts "#{greeting}"
    end
  end

  make_greeting("spanish", "Hola!")
  make_greeting("english", "Hello!")
end
```

Line: 11 Column: 4 Ruby Soft Tabs: 2 self.make_greeting(language, greeting)

Accessing the singleton meta-class explicitly

singleton_class.rb

```ruby
module Kernel
  def singleton_class
    class << self
      self
    end
  end
end
```

Line: 7 Column: 4 Ruby Soft Tabs: 2

Child classes get enhanced with class methods

```
class Application

  # specific applications get a singleton class level instances for holding
  # the array of pages contained in the application and a the home page
  def self.inherited(child) #:nodoc:
    metaclass = Utils.metaclass(child)
    Utils.attr_array(child, :pages, :create_accessor => false)
    metaclass.instance_eval do
      define_method(:homepage) { @homepage }
      define_method(:homepage=) { |page| @homepage = page }
    end
    super
  end
```

```
module Utils

  def self.metaclass(clazz = self)
    class << clazz; self; end
  end
```

Inheritance the Ruby Way

Many Ruby DSLs create class methods on subclasses using the meta-class (singleton class)

Rails uses this technique in *ActiveRecord*

```ruby
class ProjectsUser < ActiveRecord::Base
  belongs_to :project
  belongs_to :user

  ROLES = { 1 => "Worker", 2 => "Manager"}

  validates_uniqueness_of :project_id, :scope => :user_id
  validates_uniqueness_of :user_id, :scope => :project_id

  def to_s
    "#{user.login} (#{ROLES[role_type]})"
  end
end
```

A module with a “Base” class

singleton_class.rb

```ruby
module Greeters
  class Base
    def self.greet(language, greeting)
      make_greeting(language, greeting)
    end

    def self.make_greeting(language, greeting)
      singleton_class.class_eval do
        define_method "greet_in_#{language}" do
          puts "#{greeting}"
        end
      end
    end
  end
end
```

Line: 7 Column: 4 Ruby Soft Tabs: 2 << self

Methods get added to our classes with simple declarations

singleton_class.rb

```
class Person < Greeters::Base
  greet :french, "Bonjour!"
  greet :spanish, "Hola!"
  greet :english, "Hello!"
  greet :dutch, "Hallo!"
end
```

Line: 7 Column: 4 Ruby Soft Tabs: 2 <...

Getting closer to a DSL!

Now our class has a new set of class methods

Default (79,16)

```
/> irb
irb(main):001:0> require 'singleton_class.rb'
=> true
irb(main):002:0> Person.greet_in_french
Bonjour!
=> nil
irb(main):003:0> Person.greet_in_spanish
Hola!
=> nil
irb(main):004:0> Person.greet_in_english
Hello!
=> nil
irb(main):005:0> Person.greet_in_dutch
Hallo!
=> nil
irb(main):006:0> _
```


AOP
THE RUBY
WAY

With *alias_method* you can wrap an existing method...

created_on_hook.rb

```ruby
class Object
  attr_accessor :created_on
end

class Class
  alias_method :old_new, :new
  def new(*args)
    instance = old_new(*args)
    instance.created_on = Time.now
    instance
  end
end
```

Line: 9 Column: 24 Ruby Soft Tabs: 2 new(*args)

...effectively intercepting any calls and injecting any desired behavior

Default (74,8)

```
irb(main):001:0> require 'created_on_hook.rb'
=> true
irb(main):002:0> class Boo
irb(main):003:1> end
=> nil
irb(main):004:0> boo = Boo.new
=> #<Boo:0x60d3c @created_on=Thu Jan 10 15:52:33 EST 2008>
irb(main):005:0> _
```


Meta-Programming
Events

Included Hook

Ruby has a rich event model associated with static and dynamic changes of the code

included.rb

```ruby
module UberCoolUtilities
  def self.included(target)
    puts "module #{self} has been included in #{target}"
  end
end

class MyCoolClass
  include UberCoolUtilities
end
```

Line: 9 Column: 4 Ruby Soft Tabs:

Default (58,5)

```
/> irb
irb(main):001:0> require 'included.rb'
module UberCoolUtilities has been included in MyCoolClass
=> true
irb(main):002:0>
```


Method Missing

Ruby provides several hook methods that can be used to created custom behaviors

method_missing

method_added

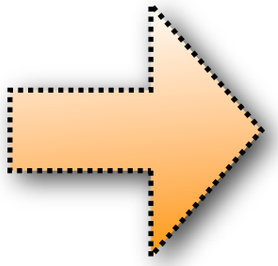
method_removed

Ruby's Markup Builder is an example of what can be achieved with method missing

```ruby
require 'rubygems'
require 'builder'

builder = Builder::XmlMarkup.new

builder.html {
  builder.head {
    builder.title "History"
  }
  builder.body {
    builder.h1 "Header"
    builder.comment! "A Comment"
  }
}
```



```html
<html>
    <head>
        <title>
            History
        </title>
    </head>
    <body>
        <h1>
            Header
        </h1>
        <!-- A Comment -->
    </body>
</html>
```

Let's implement the greeter example using *method_missing*

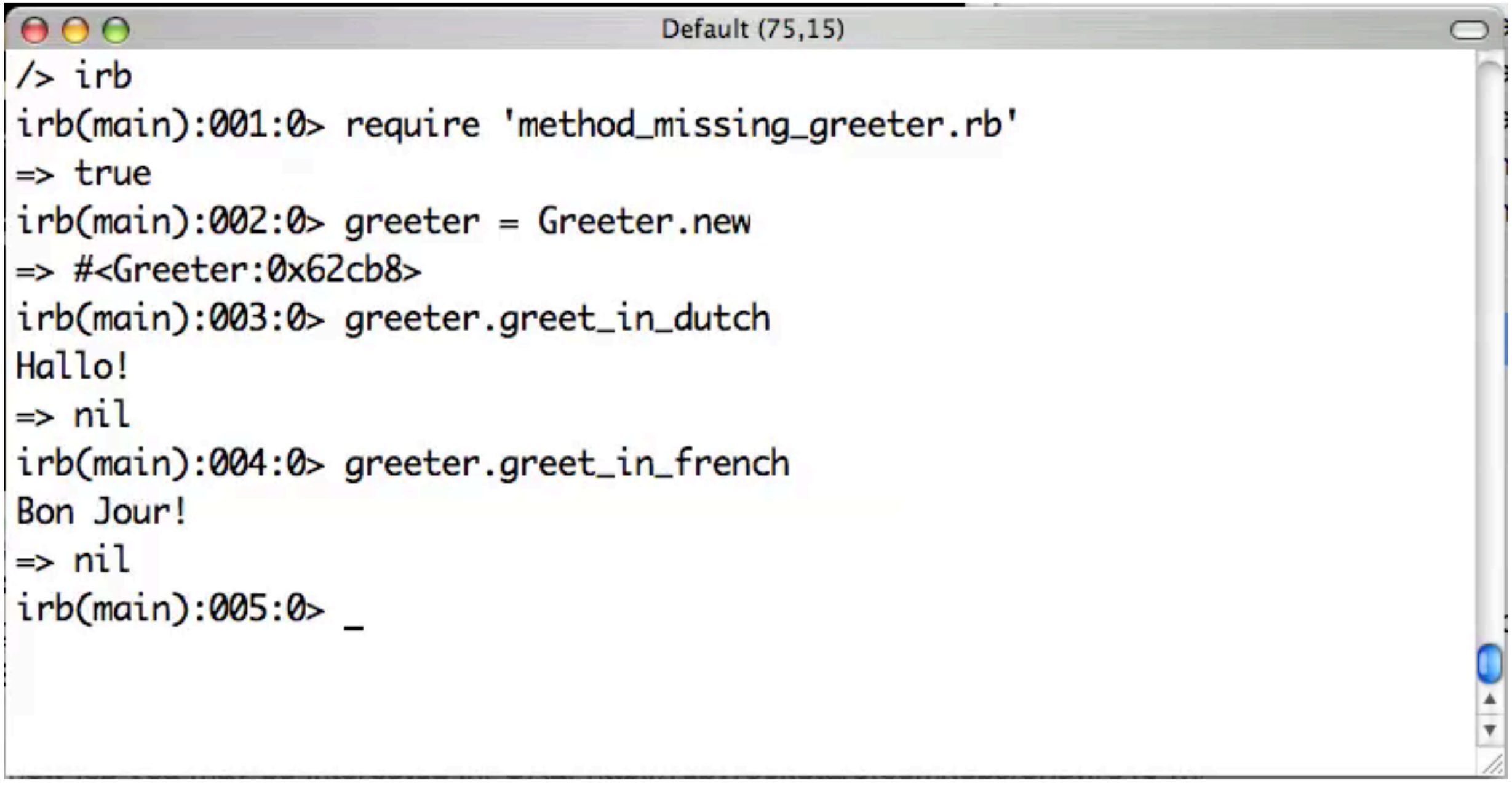
method_missing_greeter.rb

```ruby
class Greeter
  Greetings = {:english => 'Hello',
               :french => 'Bon Jour',
               :dutch => 'Hallo',
               :spanish => 'Hola'}

  def method_missing(method_name, *args)
    name = method_name.to_s.split(/_/).first
    language = method_name.to_s.split(/_/).last.to_sym
    if name == 'greet'
      puts Greetings[language] + '!'
    end
  end
end
```

Line: 15 Column: 1 Ruby Soft Tabs: 2 method_missing(method_name...

greeter using *method_missing*



I was bored...

In the Ruby world I've worked with Rails and Merb

...thinking of web apps in terms of request & response

meanwhile... in the Java world I was working with Tapestry and Wicket

...thinking about Pages, Components and Events

Started to hack to see what it would be to build something like Tapestry/Wicket in **Ruby**

...actually it all started as an learning exercise

...part of a conversation among friends about the need for IoC and AOP frameworks for dynamic languages

...but then I kept going

...somehow I ended building component-oriented web framework in Ruby

You might be asking yourself, but why?

Why not?

TRELLIS

COMPONENT-DRIVEN WEB DEVELOPMENT IN RUBY

My goals:

✓ Magic like Rails; less code more action

✓ Small like Camping... a micro-framework

✓ Components like Tapestry/Wicket

✓ Clean HTML templates...

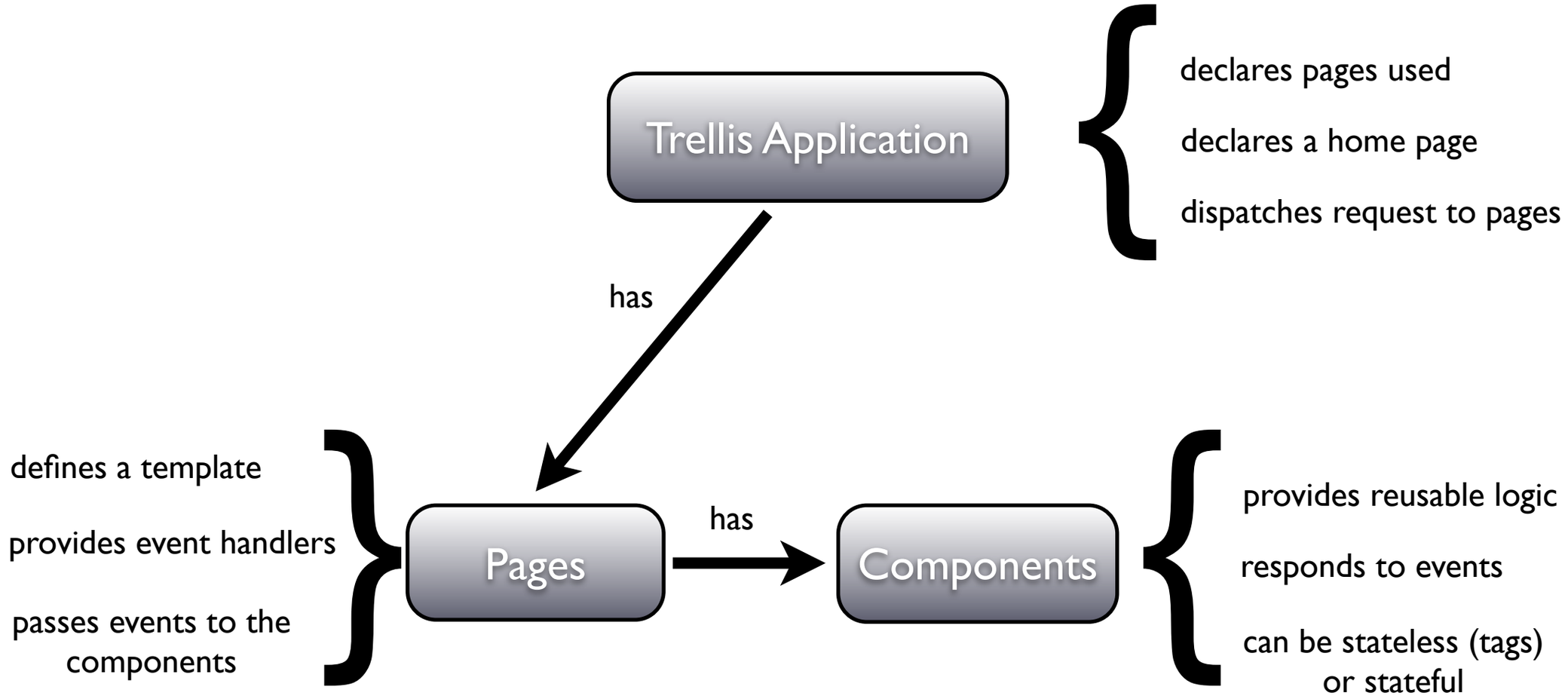
✓ ...but also programatic HTML generation

✓ Non-managed components, no pooling

TRELLIS

COMPONENT-DRIVEN WEB DEVELOPMENT IN RUBY

The big picture:



TRELLIS

COMPONENT-DRIVEN WEB DEVELOPMENT IN RUBY

The simplest **Trellis** App, one Application and one Page

```ruby
require 'trellis/trellis'

include Trellis

module Simplest

  class Simplest < Application
    home :start
  end

  class Start < Page
    attr_accessor :current_time

    def initialize
      @current_time = Time.now
    end
  end

  Simplest.new.start
end
```

The Code

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>Simplest Trellis Application</title>
    </head>
    <body>
        <h1>Simplest Start Page</h1>

        <p>This is the start page for this application, a good place to start your modifications.
            Just to prove this is live: </p>

        <p>The current time is: <trellis:value name="current_time">${currentTime}.</trellis:value> </p>

        <p>[<trellis:page_link tpage="start">refresh</trellis:page_link>]</p>
    </body>
</html>
```

The Template

What did I use to put this thing together...

➡A boat load of meta-programming

➡Rack ...HTTP the Ruby way

➡Radius ...XML tags for the templates

➡Builder ...for building HTML/XML

➡Hpricot, REXML ...parsing searching HTML/XML

➡Paginator ...for duh, pagination

➡Parts of Rails ...so far ActiveSupport (Inflectors)

Let's do some code spelunking on the **Trellis** codebase...

yes, there are some things that I'm not proud of but...

...this is pre-alpha, unreleased v0.000001 ;-)

Some of the example applications that I've put together in order of complexity

- hilo
- guest_book
- hangman
- yui
- stateful_counters
- flickr
- simple_blog
- crud_components


WHAT HAVE
WE LEARNED?

There is more to **Ruby** than Rails!

Building a Framework?
Give **Ruby** a try!

Build the language up towards the problem