Introduction to Rango

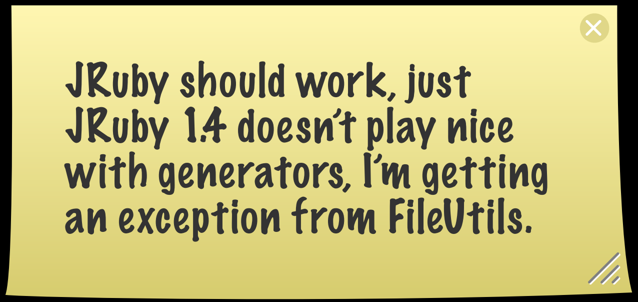
Jakub Šťastný aka @botanicus

Lead Developer at Headshift

What is Rango?

- Rack-based light-weight web framework
- As agnostic and flexible as possible
- *The goal is to provide solid framework for building sites in Ruby. More robust than Sinatra, but smaller than Rails or Merb. It should be good for apps of all sizes, from small web services to big sites.*

Features


JRuby should work, just JRuby 1.4 doesn't play nice with generators, I'm getting an exception from FileUtils.

- Ruby 1.9.1, 1.9.2, JRuby (jruby --1.9)
- Provides generators, but don't insist on a certain structure.
- Code reloading by shotgun (*via generators*)
- Bundler support (*via generators*)
- Inspired by Django & Merb

- Django terminology: *View is called template and view means an action in a controller.*
- You can use just whichever subset of Rango you want and it will work.
- Any strict conventions, just recommendations, often use hooks or strategies.

Boot Process

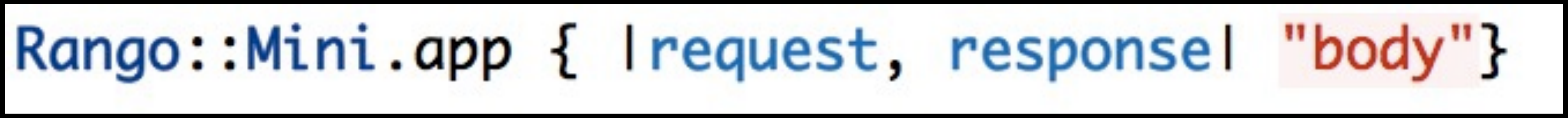
- Script init.rb should setup environment.
- Work as a runner: ./init.rb myscript.rb
- And as a console: ./init.rb -i
- Or both altogether: ./init.rb -i myscript.rb
- Start web server: ./config.ru
- Rewrite arguments: ./config.ru -s thin

Routers

- Usher, Rack-mount
- URLMap, Rack-router
- Rango::UrlHelper for url(:post, id)
- env[“rango.router.params”]

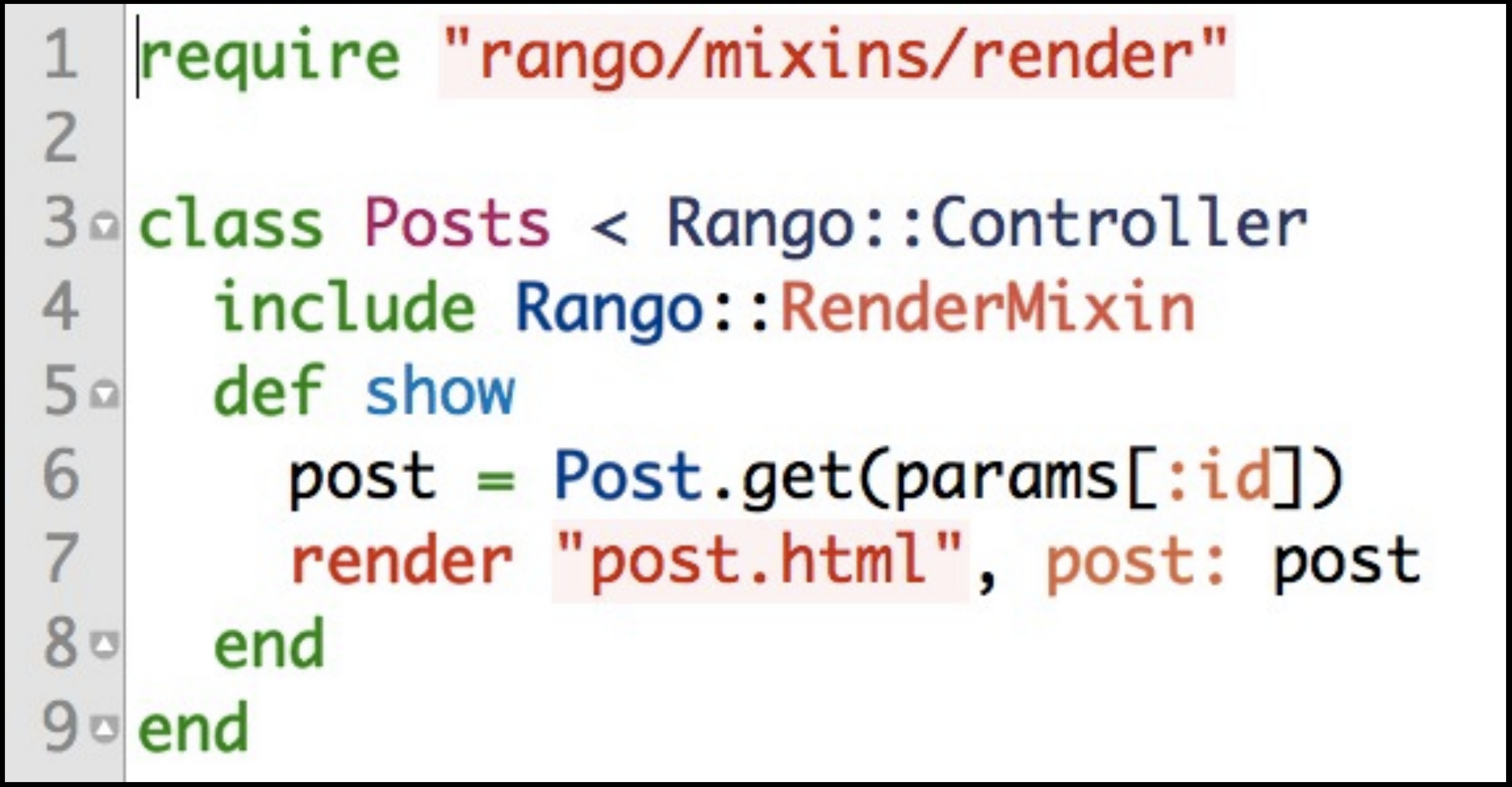
Rendering

- You don’t need controllers for it:

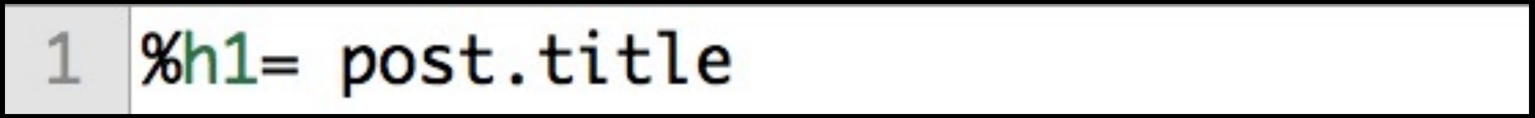


```
Rango::Mini.app { |request, response| "body"}
```

RenderMixin



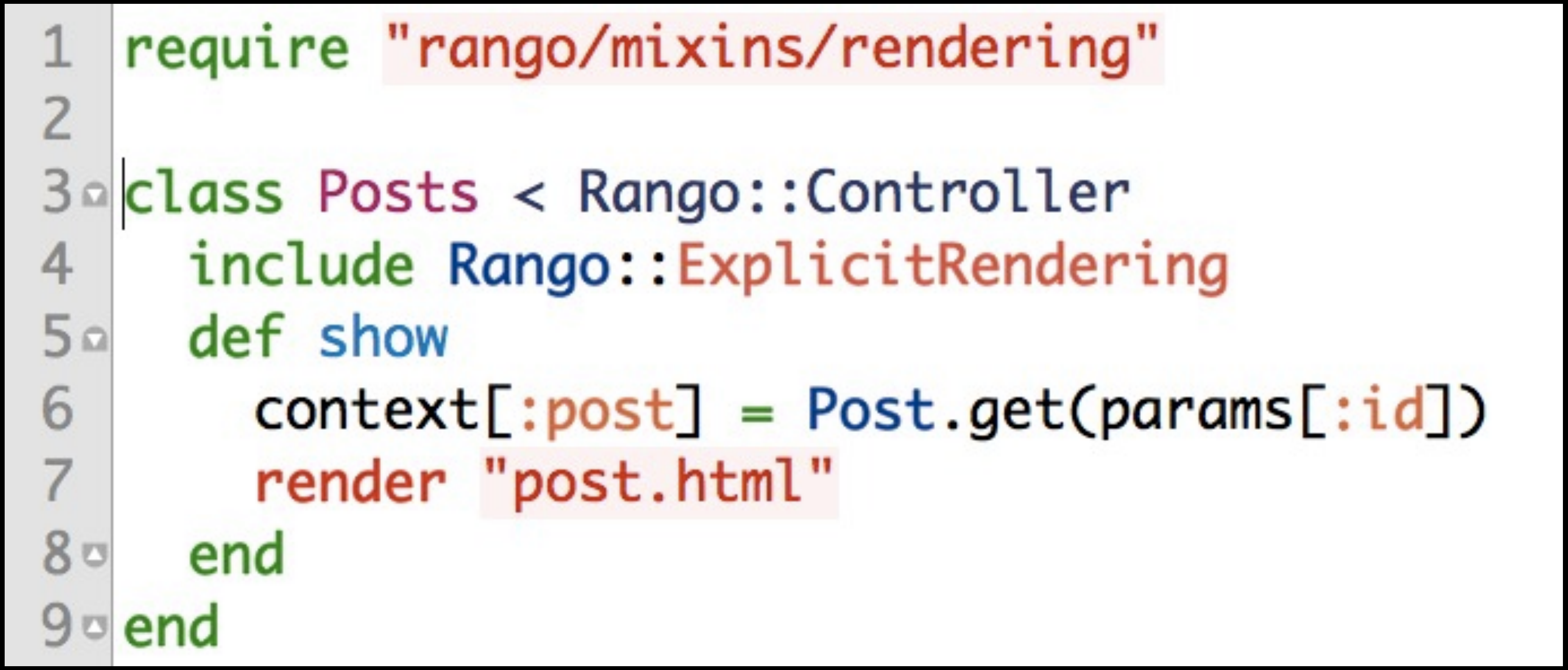
```ruby
require "rango/mixins/render"

class Posts < Rango::Controller
  include Rango::RenderMixin
  def show
    post = Post.get(params[:id])
    render "post.html", post: post
  end
end
```



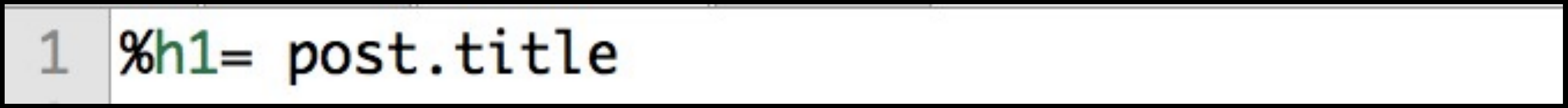
```haml
%h1= post.title
```

- The most low-level rendering layer.
- Template will be evaluated in context of the third argument which defaults to Object.new.
- post is defined as a local variable.

ExplicitRendering



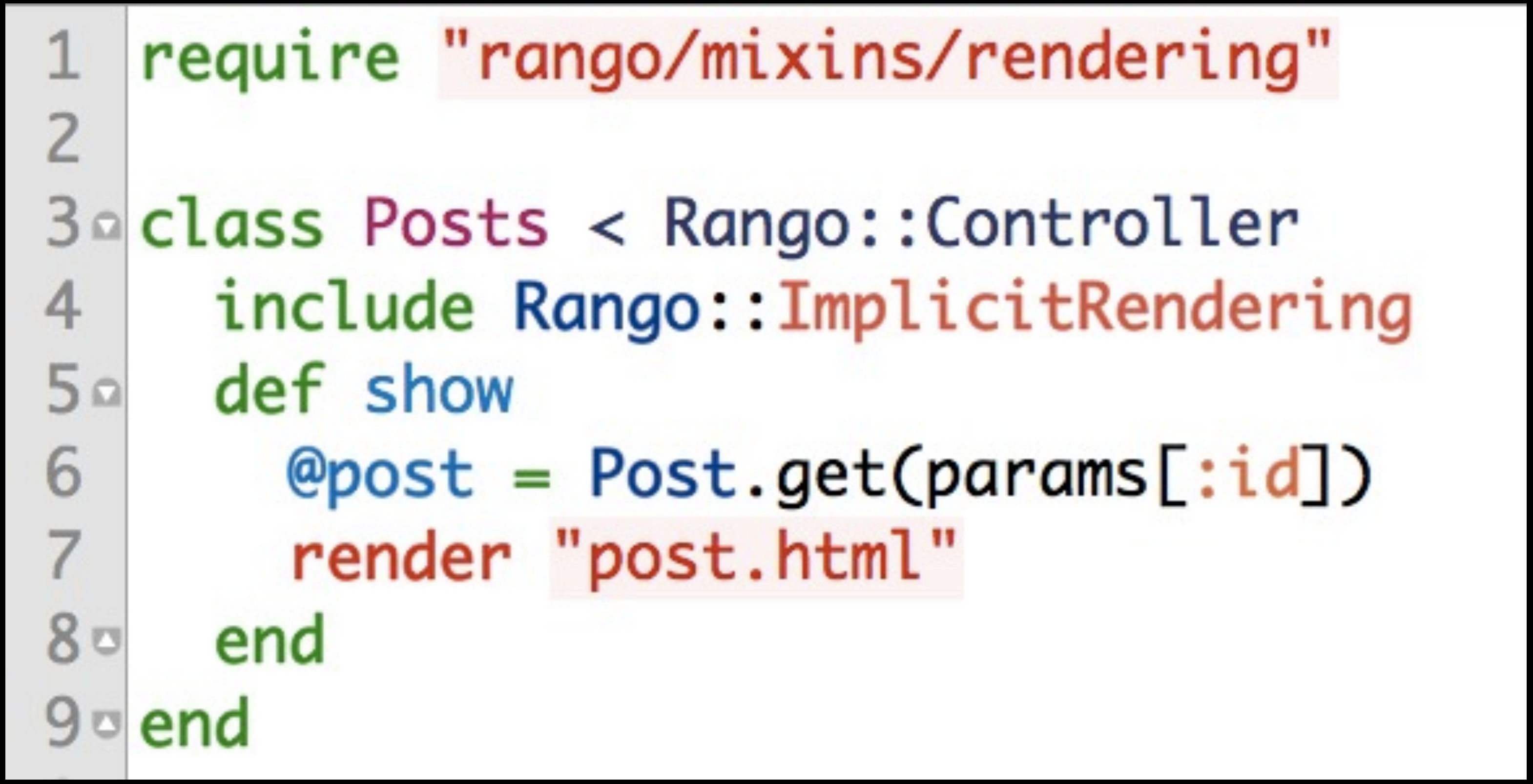
```ruby
require "rango/mixins/rendering"

class Posts < Rango::Controller
  include Rango::ExplicitRendering
  def show
    context[:post] = Post.get(params[:id])
    render "post.html"
  end
end
```



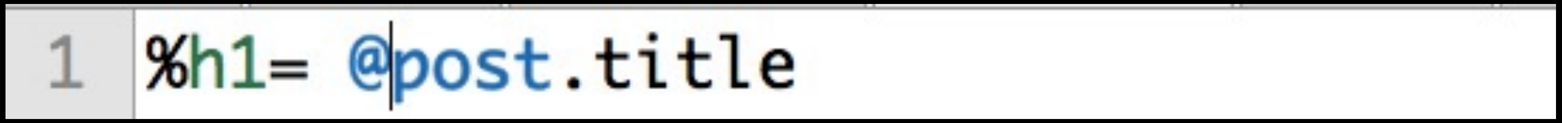
```haml
%h1= post.title
```

- Template will be evaluated in context of self.scope which you can redefine.
- Data from self.context hash will be available as local variables.
- Context defaults to {request: self.request}.

ImplicitRendering



```ruby
require "rango/mixins/rendering"

class Posts < Rango::Controller
  include Rango::ImplicitRendering
  def show
    @post = Post.get(params[:id])
    render "post.html"
  end
end
```



```haml
%h1= @post.title
```

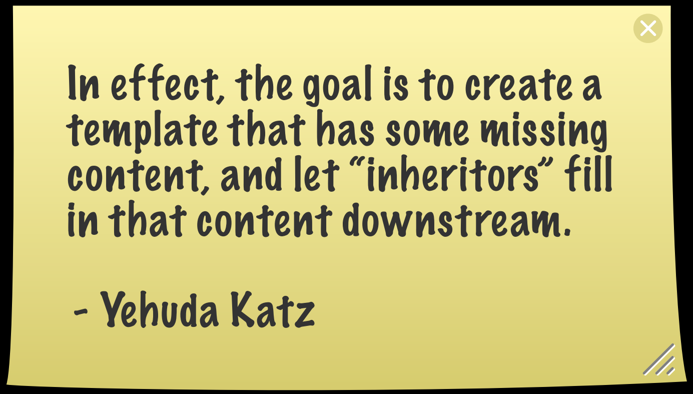
- Standard Rails-like rendering.
- Template will be evaluated in context of self, so you can use.
- Methods as self.request or instance variables as @post are evaluated on self directly, no context required.

Template Inheritance

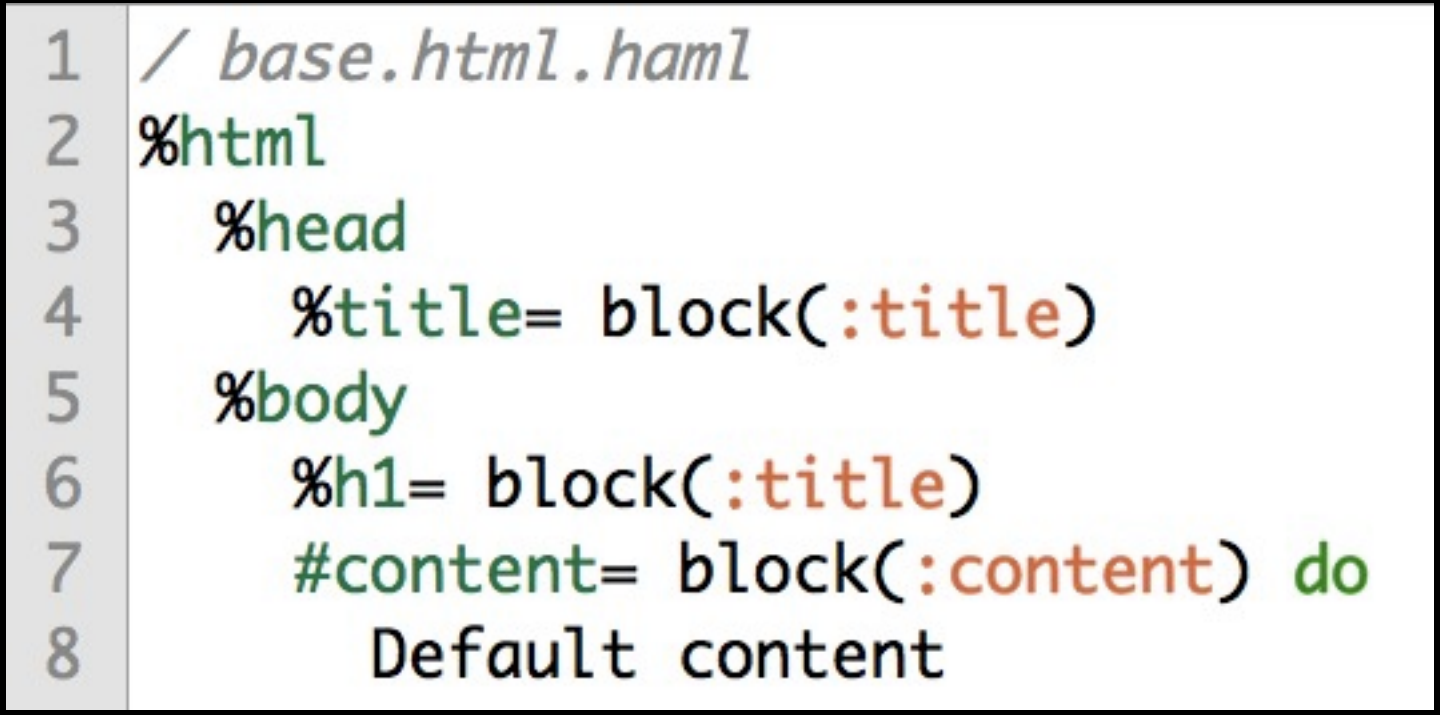
- Much more flexible and cleaner than layout/view, no hacking around with content_for and a bunch of partials.
- Just an inheritance: (*admin/posts.html* => *admin/base.html* => *base.html*).
- Rango has very good and powerful implementation.

- Rango comes first, but @hassox already used for his Pancake and it seems a lot of people want it in Merb.

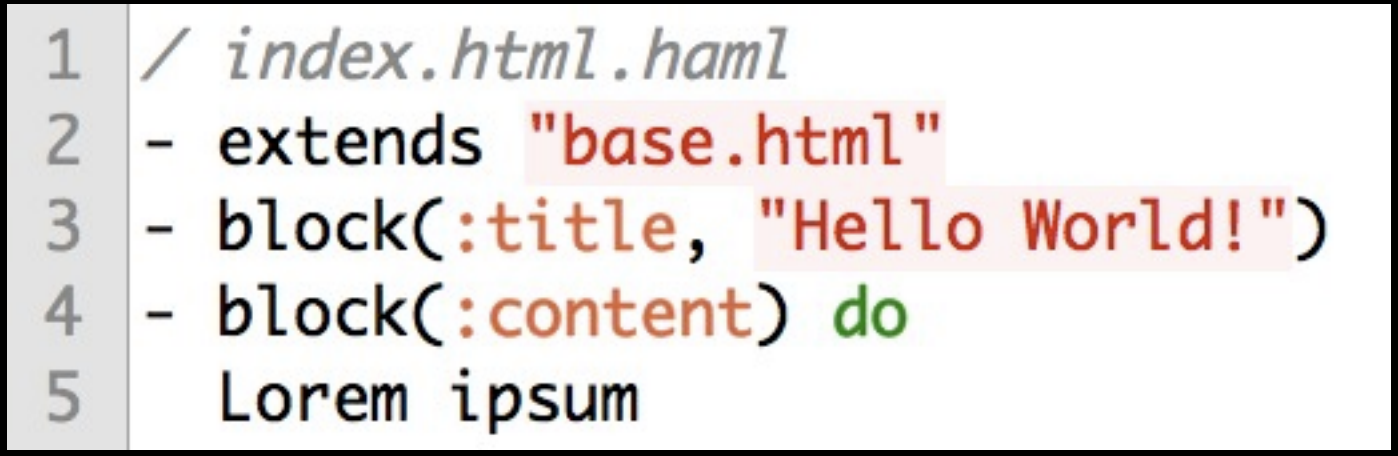
Basic Example


In effect, the goal is to create a template that has some missing content, and let "inheritors" fill in that content downstream.

- Yehuda Katz


```
1 / base.html.haml
2 %html
3   %head
4     %title= block(:title)
5   %body
6     %h1= block(:title)
7     #content= block(:content) do
8       Default content
```

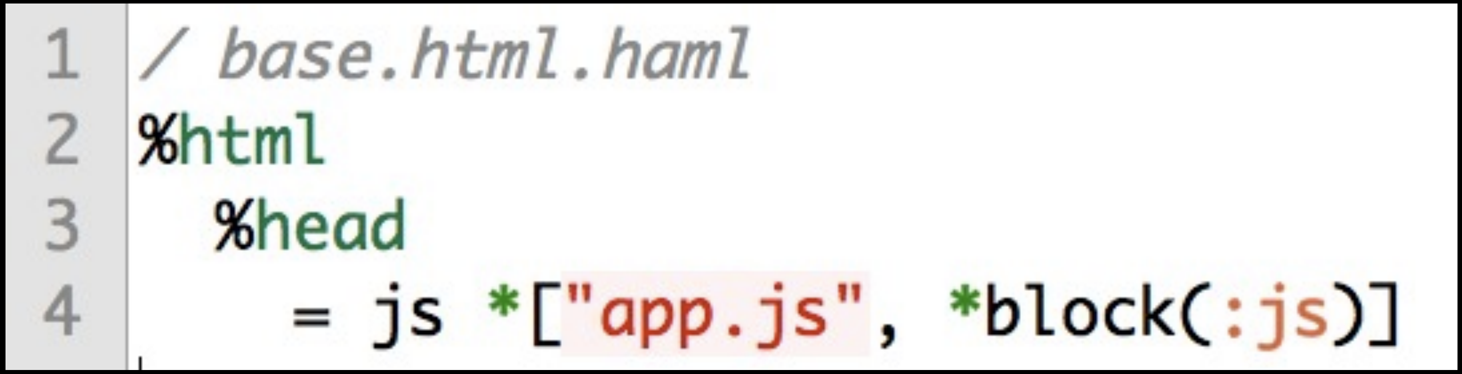
#8 Default value for block


```
1 / index.html.haml
2 - extends "base.html"
3 - block(:title, "Hello World!")
4 - block(:content) do
5   Lorem ipsum
```

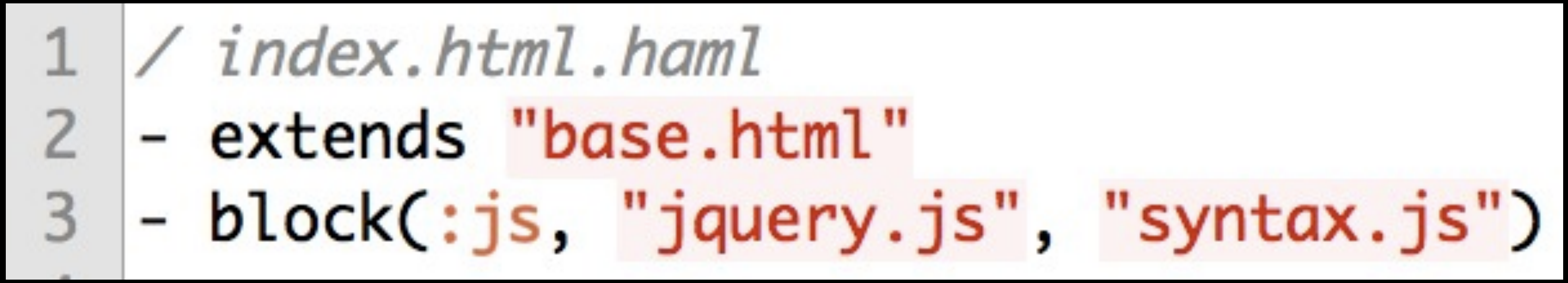
- render "index.html"
- extends "base.html" will parse base.html and replace its blocks by blocks in index.html

Returning values

- Block can return whichever value via arguments.
- Block returns captured string from &block.

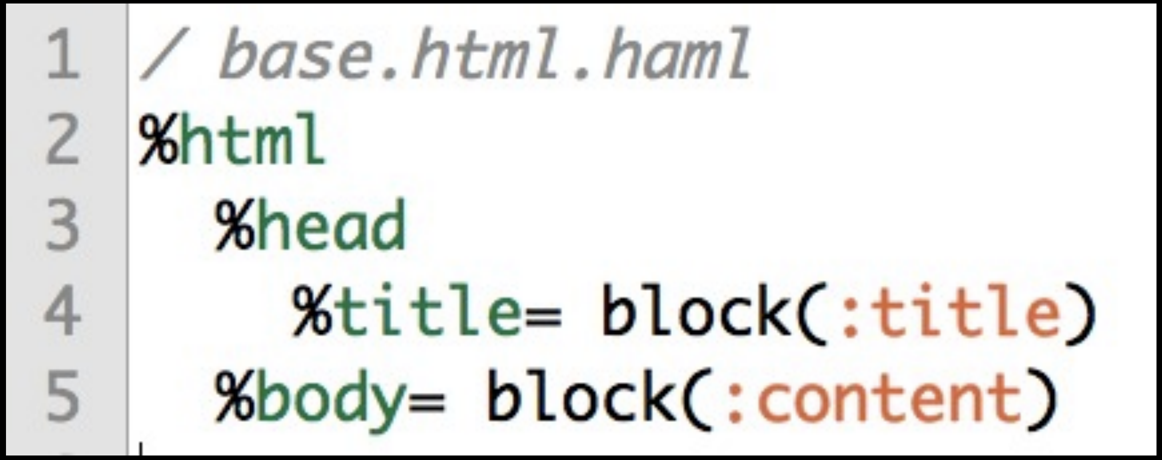


```haml
/ base.html.haml
%html
  %head
    = js *["app.js", *block(:js)]
```

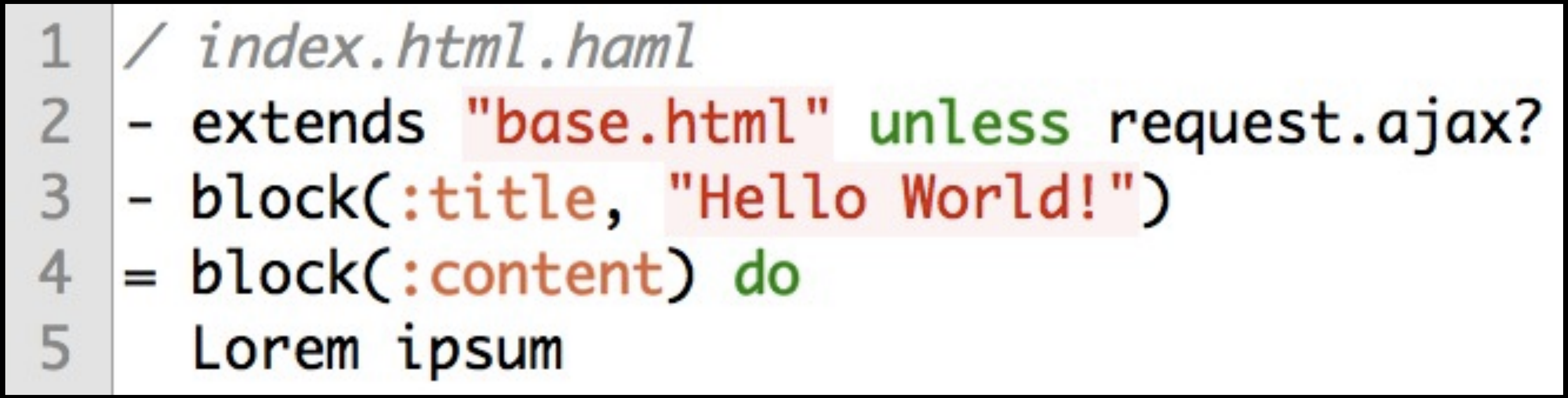


```haml
/ index.html.haml
- extends "base.html"
- block(:js, "jquery.js", "syntax.js")
```

AJAX



```haml
/ base.html.haml
%html
  %head
    %title= block(:title)
  %body= block(:content)
```



```haml
/ index.html.haml
- extends "base.html" unless request.ajax?
- block(:title, "Hello World!")
= block(:content) do
  Lorem ipsum
```

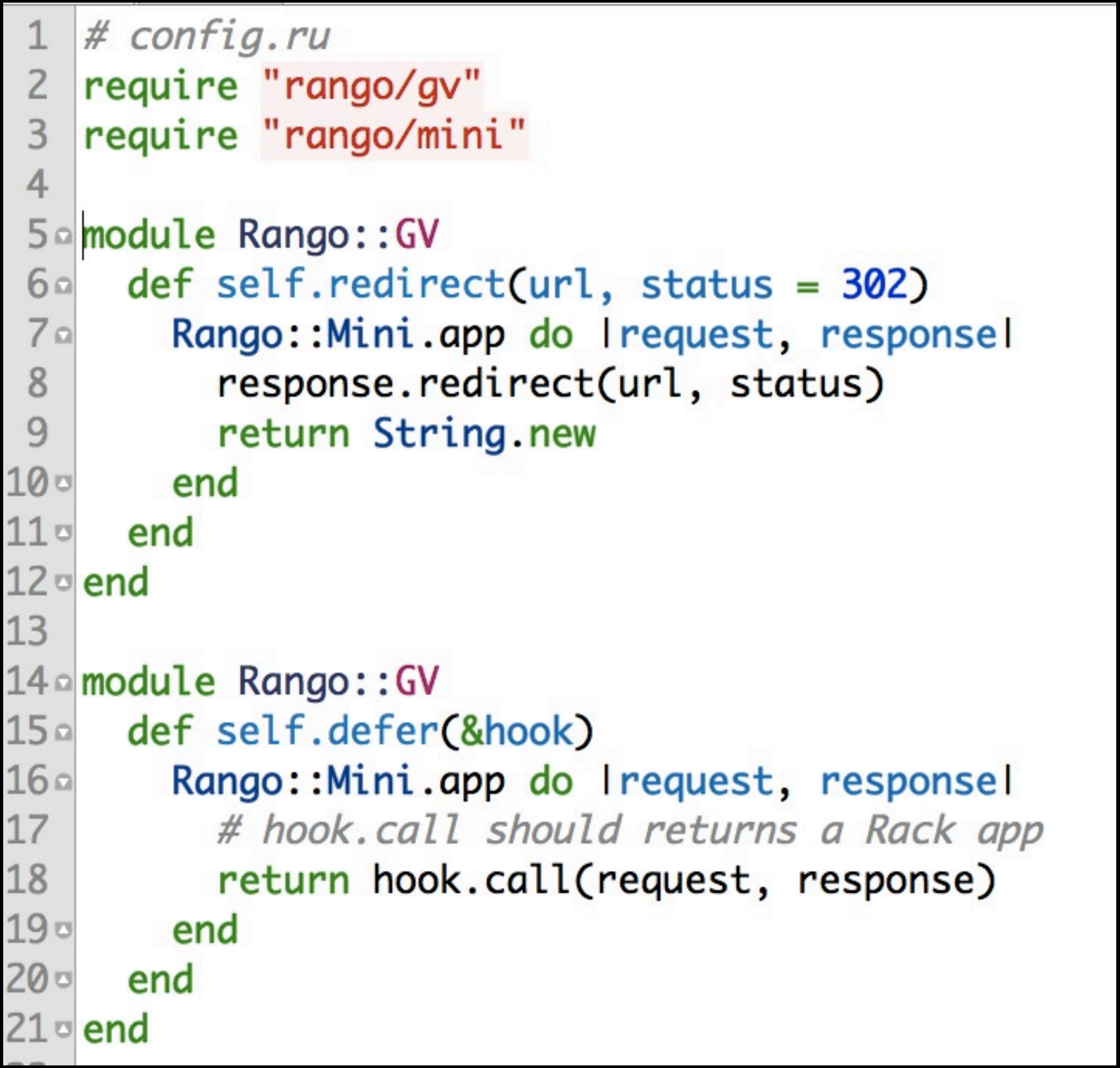
- It will render normal page with layout if the request isn't AJAX.
- If request is AJAX it will just render index.html. Haml render everything starting with =, it will simly render content block which we can use for updating part of site via JS.

Generic Views (GV)

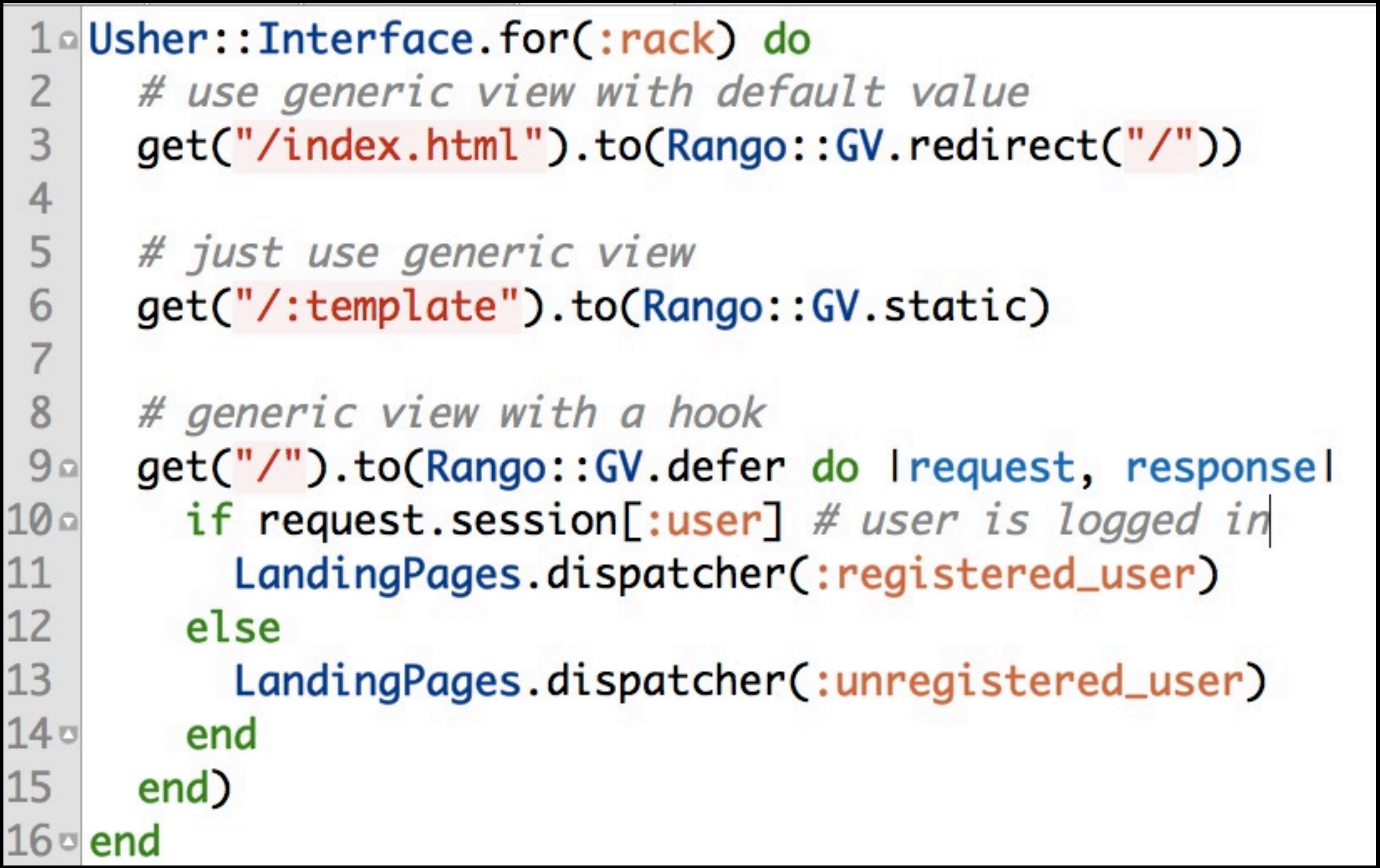
- Borrowed from Django
- Just a reusable Rack app
- Can extend router (redirect, defer)
- Just render template Rango::GV.static
- Scaffolding

GV: Router Extensions

- Deffered routes: check request and decide where to route (different actions for desktop/mobile, for registered/unregistered users etc).
- Rango::GV.redirect may be useful if your router doesn’t support it.

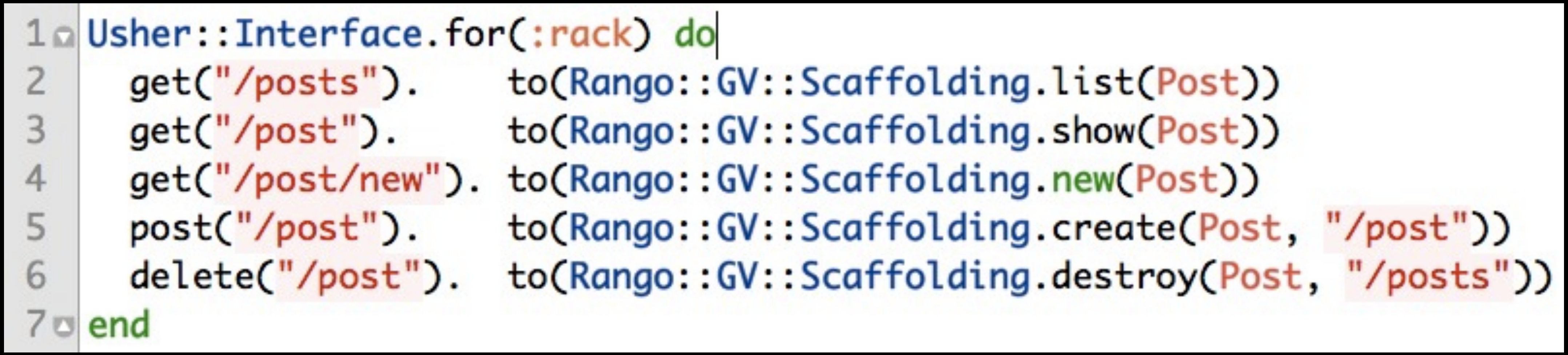


```ruby
# config.ru
require "rango/gv"
require "rango/mini"

module Rango::GV
  def self.redirect(url, status = 302)
    Rango::Mini.app do |request, response|
      response.redirect(url, status)
      return String.new
    end
  end
end

module Rango::GV
  def self.defer(&hook)
    Rango::Mini.app do |request, response|
      # hook.call should returns a Rack app
      return hook.call(request, response)
    end
  end
end
```



```ruby
Usher::Interface.for(:rack) do
  # use generic view with default value
  get("/index.html").to(Rango::GV.redirect("/"))

  # just use generic view
  get("/:template").to(Rango::GV.static)

  # generic view with a hook
  get("/").to(Rango::GV.defer do |request, response|
    if request.session[:user] # user is logged in
      LandingPages.dispatcher(:registered_user)
    else
      LandingPages.dispatcher(:unregistered_user)
    end
  end)
end
```

Scaffolding via GV



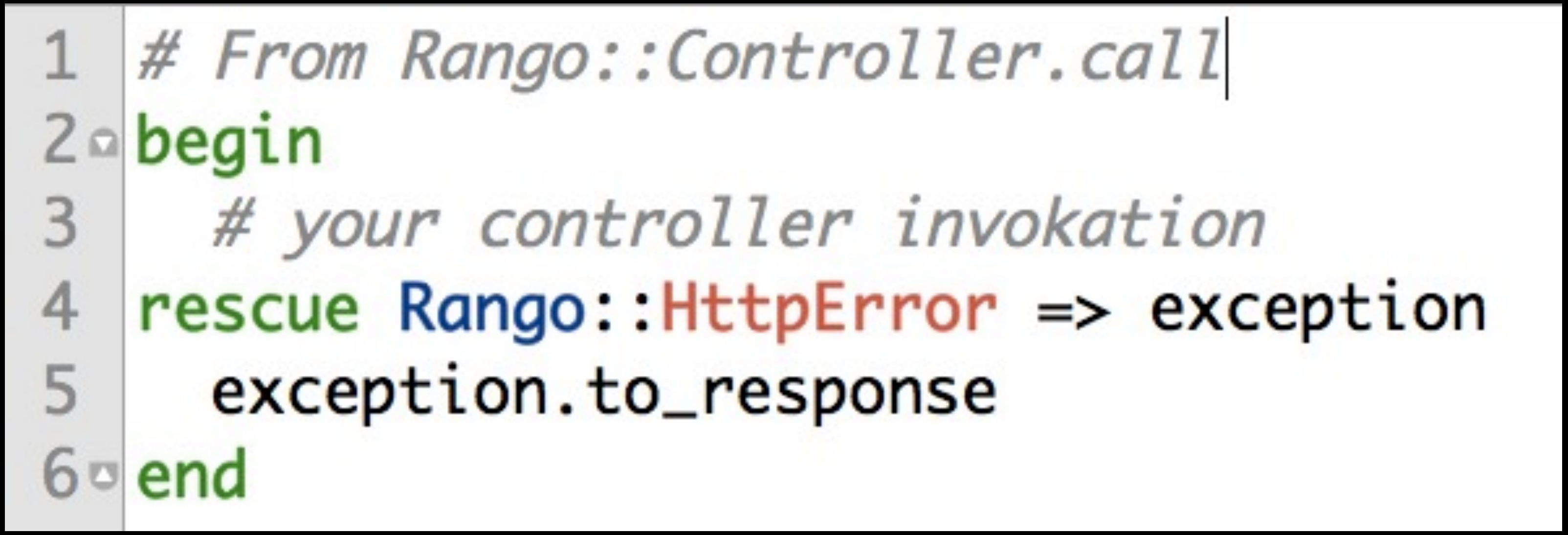
```
Usher::Interface.for(:rack) do
  get("/posts").    to(Rango::GV::Scaffolding.list(Post))
  get("/post").     to(Rango::GV::Scaffolding.show(Post))
  get("/post/new"). to(Rango::GV::Scaffolding.new(Post))
  post("/post").    to(Rango::GV::Scaffolding.create(Post, "/post"))
  delete("/post").  to(Rango::GV::Scaffolding.destroy(Post, "/posts"))
end
```

- *Coming soon, not committed yet.*
- Will be much easier if there would be generic interface for resources in routers.
- Customization: create /scaffolding/list.html.haml etc in your templates path.

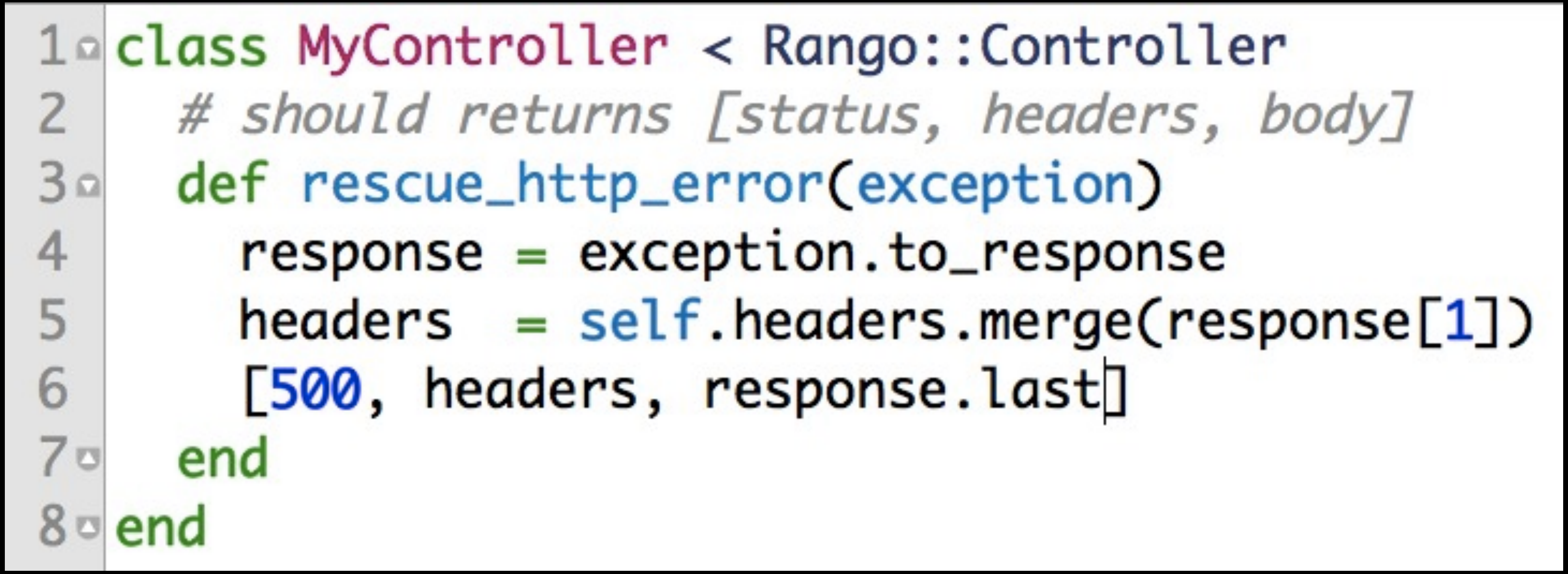
HTTP Errors Handling

- Based on exceptions inherited from Rango::HttpError.
- Error404 resp. NotFound.
- Has status, content_type and headers.

Rango::Controller#rescue_http_error

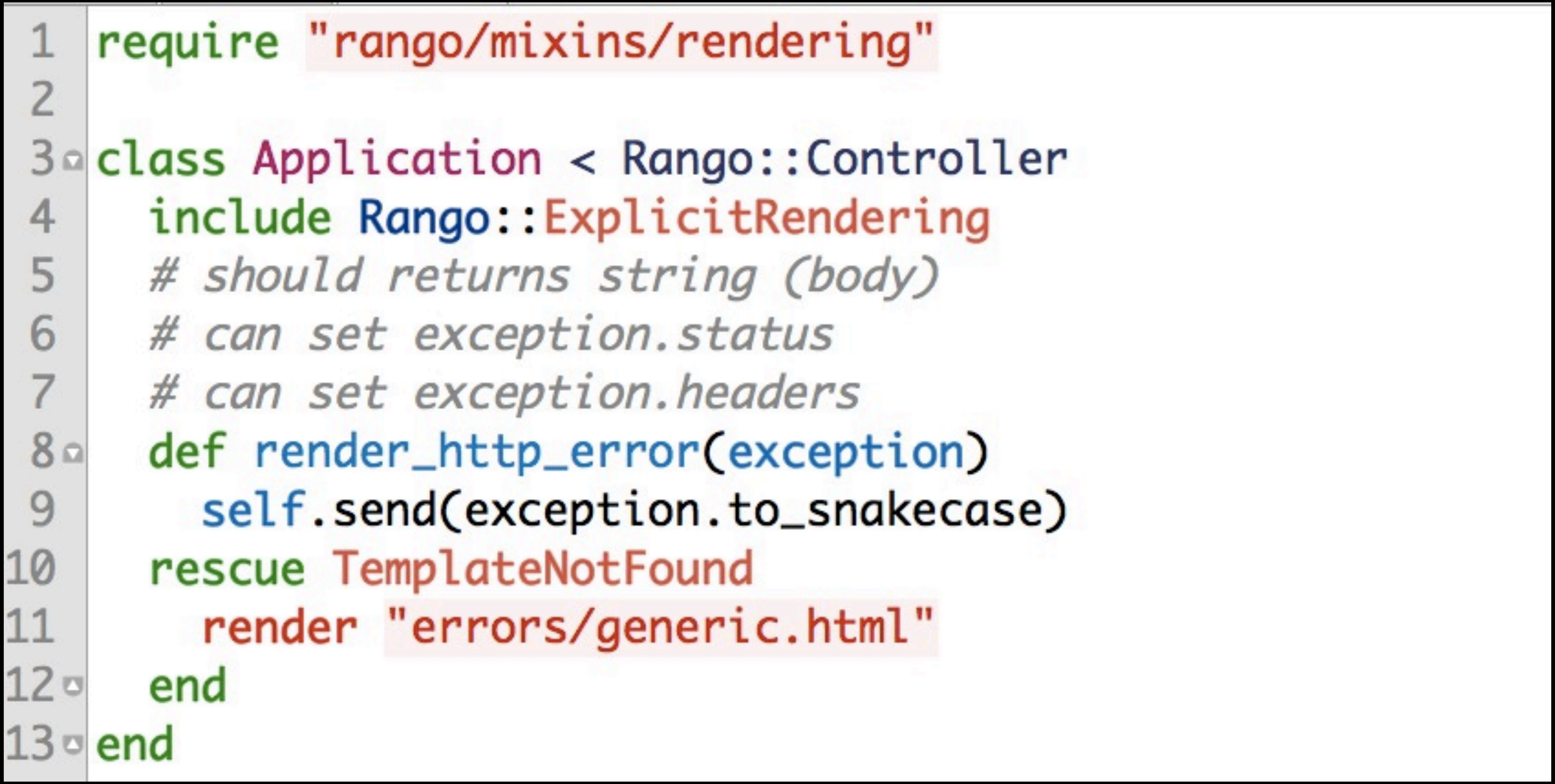


```ruby
# From Rango::Controller.call
begin
  # your controller invokation
rescue Rango::HttpError => exception
  exception.to_response
end
```



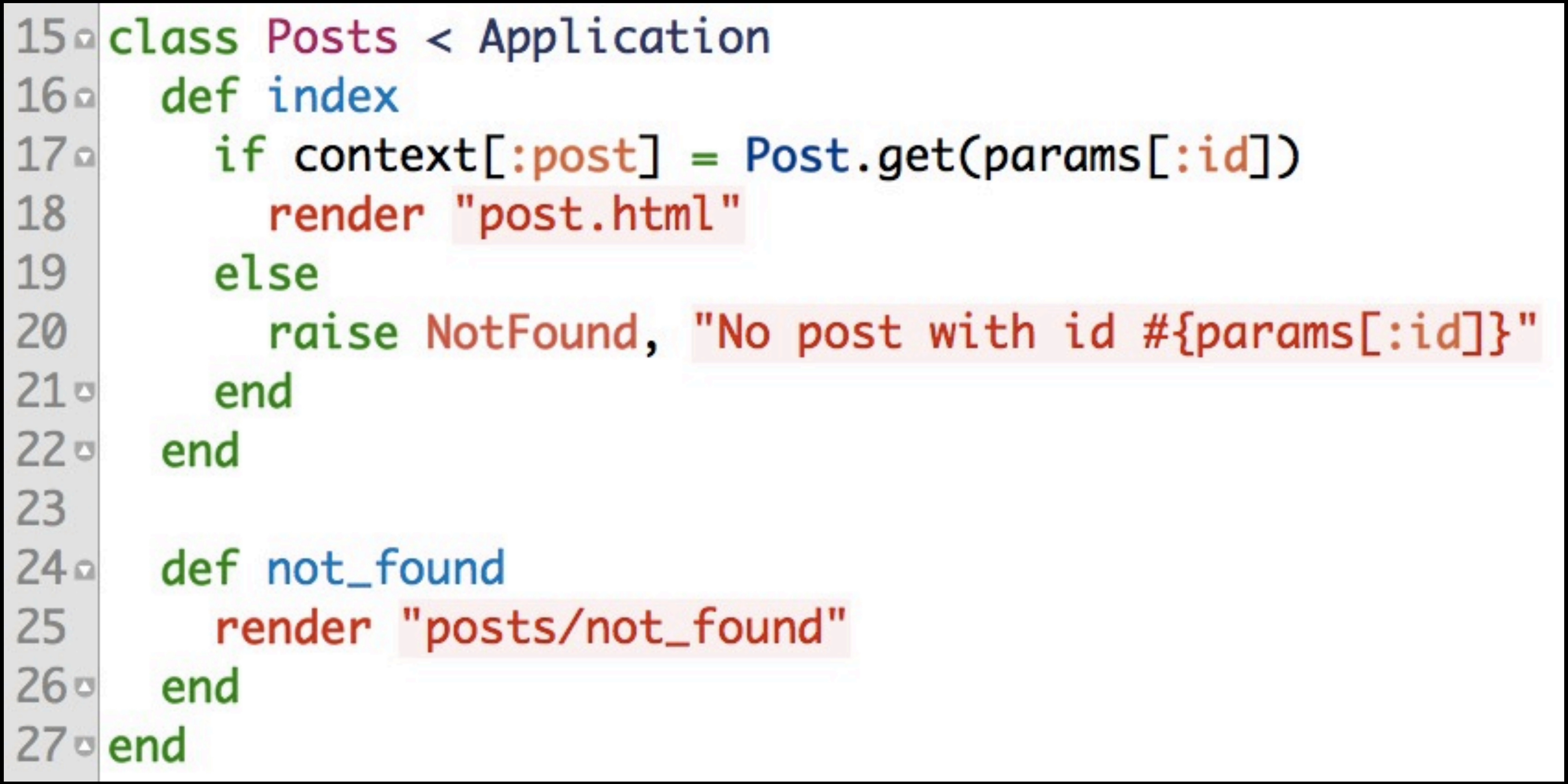
```ruby
class MyController < Rango::Controller
  # should returns [status, headers, body]
  def rescue_http_error(exception)
    response = exception.to_response
    headers  = self.headers.merge(response[1])
    [500, headers, response.last]
  end
end
```

Rango::Controller#render_http_error 1



```ruby
require "rango/mixins/rendering"

class Application < Rango::Controller
  include Rango::ExplicitRendering
  # should returns string (body)
  # can set exception.status
  # can set exception.headers
  def render_http_error(exception)
    self.send(exception.to_snakecase)
  rescue TemplateNotFound
    render "errors/generic.html"
  end
end
```

Rango::Controller#render_http_error 2



```ruby
class Posts < Application
  def index
    if context[:post] = Post.get(params[:id])
      render "post.html"
    else
      raise NotFound, "No post with id #{params[:id]}"
    end
  end

  def not_found
    render "posts/not_found"
  end
end
```

Future

- January 2010 in Rango 0.3: Mountable apps via Pancake: each app can run standalone or be used as a library (can be distributed as gems with assets & templates)

Links

- Wiki: http://wiki.github.com/botanicus/rango
- Issues: http://github.com/botanicus/rango/issues
- Blog: http://botanicus.github.com/rango
- Twitter: http://twitter.com/RangoProject
- http://groups.google.com/group/rango-project
- http://github.com/botanicus/ruby-manor-rango

Thank you for your attention!