COM 1001
Introduction to Software Engineering

Professor Phil McMinn

An Introduction to Web Application Development

Sinatra Basics: Routes, Blocks, and HTTP

Sinatra

A **Domain Specific Language (DSL)** for writing web applications in Ruby

It is meant to be easy to learn

It is designed so that developers can write web applications quickly

It has been used commercially by several organisations from Apple, to LinkedIn, to GitHub, to the NSA.

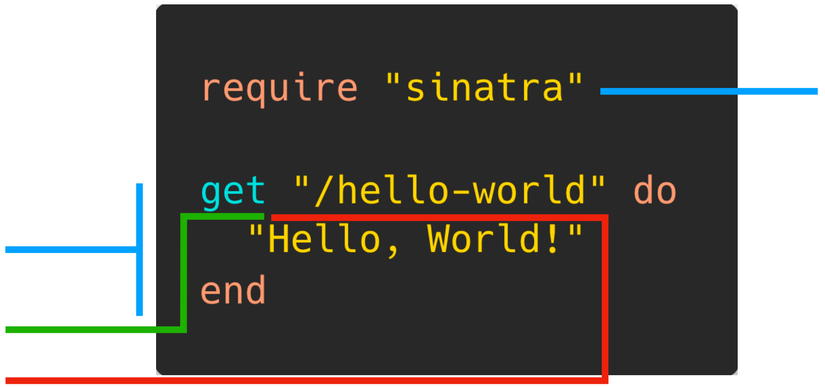
Website and documentation: `http://sinatrarb.com`

Running a Hello World example in Sinatra

(You can see this again after the lecture by watching the video on the “Getting Started” started page on Blackboard.)

Hello, World!

Sinatra involves writing blocks of Ruby code that determine which **HTTP Requests** the web application will respond to. These involve specifying a **HTTP method** and a **URL-matching pattern**, which together are called a **route**.



`getting_started/hello_world.rb`

The **URL-matching pattern** is *almost* the same thing as the **resource identifier** of a HTTP request, except the pattern can actually be used to specify more than one resource. (More on this later in the module.) In this case, the pattern maps to exactly one resource – "`/hello-world`" so the pattern and the resource identifier are essentially the *same* in this instance.

Ruby uses underscores as filename separators, but **URL-matching patterns should always use hyphens**. This might be confusing at first, but it means that the resulting URLs have hyphens rather than underscores too. This is important because it's hard to distinguish underscores from spaces when a URL is underlined. Google recommends hyphens as separators in URLs for this reason. This is known as "**kebab-case**".

The Base URL

The **Base URL** forms the initial part of every URL of your web application.

It is made up of the **HTTP protocol** being used, the **domain name** of your web server, and the **port number** it is using for your web application, as follows:

`http://localhost:4567` ⇢ **This how it usually looks, if you were developing on your own machine**

`https://box-name-4567.codio.io` ⇢ **if you're developing with Codio (recommended)**

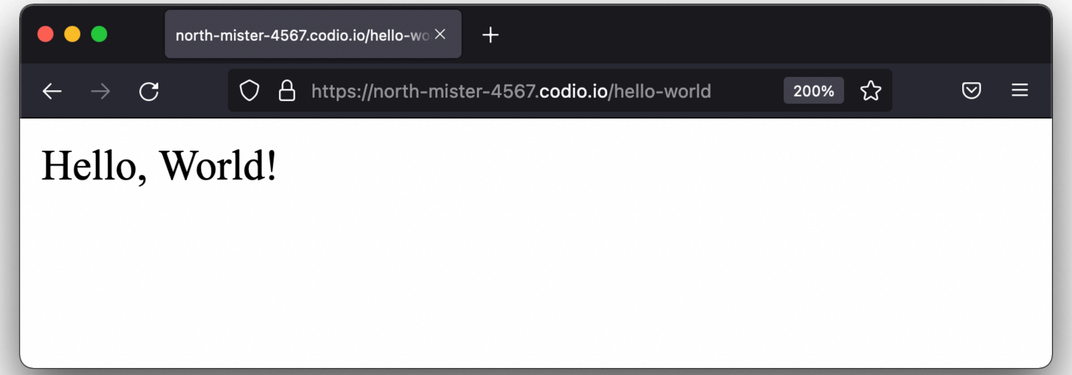
You don't need to supply the port number if the default HTTP port (80) is being used. Since we tend to use port 80 for *deployed* web applications (as opposed to those being *developed*), you often don't see the port number in normal web usage.

Viewing the Hello World Example

To invoke a Sinatra `get` route, we need to suffix the **Base URL**
with a string that matches the route's pattern,
then type the whole URL into a browser

For the Hello World example, we need to add `/hello-world`

```
require "sinatra"

get "/hello-world" do
  "Hello, World!"
end
```



My box name is "north-mister", so my URL is
https://north-mister-4567.codio.io/hello-world
Your box will have a different name, so your URL will be different of course!

The HTTP Request from My Browser

```
GET /hello-world HTTP/2
Host: north-mister-4567.codio.io
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.16; rv:84.0) Gecko/20100101 Firefox/84.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
…
```

The HTTP Response from the Web Server

```
HTTP/2 200 OK
content-type: text/html;charset=utf-8
content-length: 13
…
```

Sinatra Blocks

Sinatra **blocks** contain Ruby code specific to your web application

The block should return a string. The string forms the **content** of the **HTTP response** (i.e., a HTML web page) for the **HTTP request** that invokes the verb and route. As with Ruby methods, the `return` keyword is optional on the last line of the block.

```ruby
require "sinatra"

TIMES_TABLE = 3
LIMIT = 10

get "/times-table" do
  title = "#{TIMES_TABLE} times table"
  output = "<html><head><title>#{title}</title></head>"
  output += "<body><h1>#{title}</h1><ul>"
  (1..LIMIT).each do |i|
    result = i * TIMES_TABLE
    text = "#{i} times #{TIMES_TABLE} = #{result}"
    output += "\n\t\t\t<li>#{text}</li>"
  end
  output += "</ul></body></html>"
  output
end
```

basics/times_table.rb

Multiple Routes and Blocks

```ruby
require "sinatra"

get "/first-route" do
  "This code is run when first-route is invoked"
end

get "/second-route" do
  "This code is run when second-route is invoked"
end
```

basics/multiple_routes.rb

Suppose the base URL of your web application is https://your-box-4567.codio.io
The two blocks above would be executed by the following URLs:

https://your-box-4567.codio.io/first-route
https://your-box-4567.codio.io/second-route

The Effect of Using puts

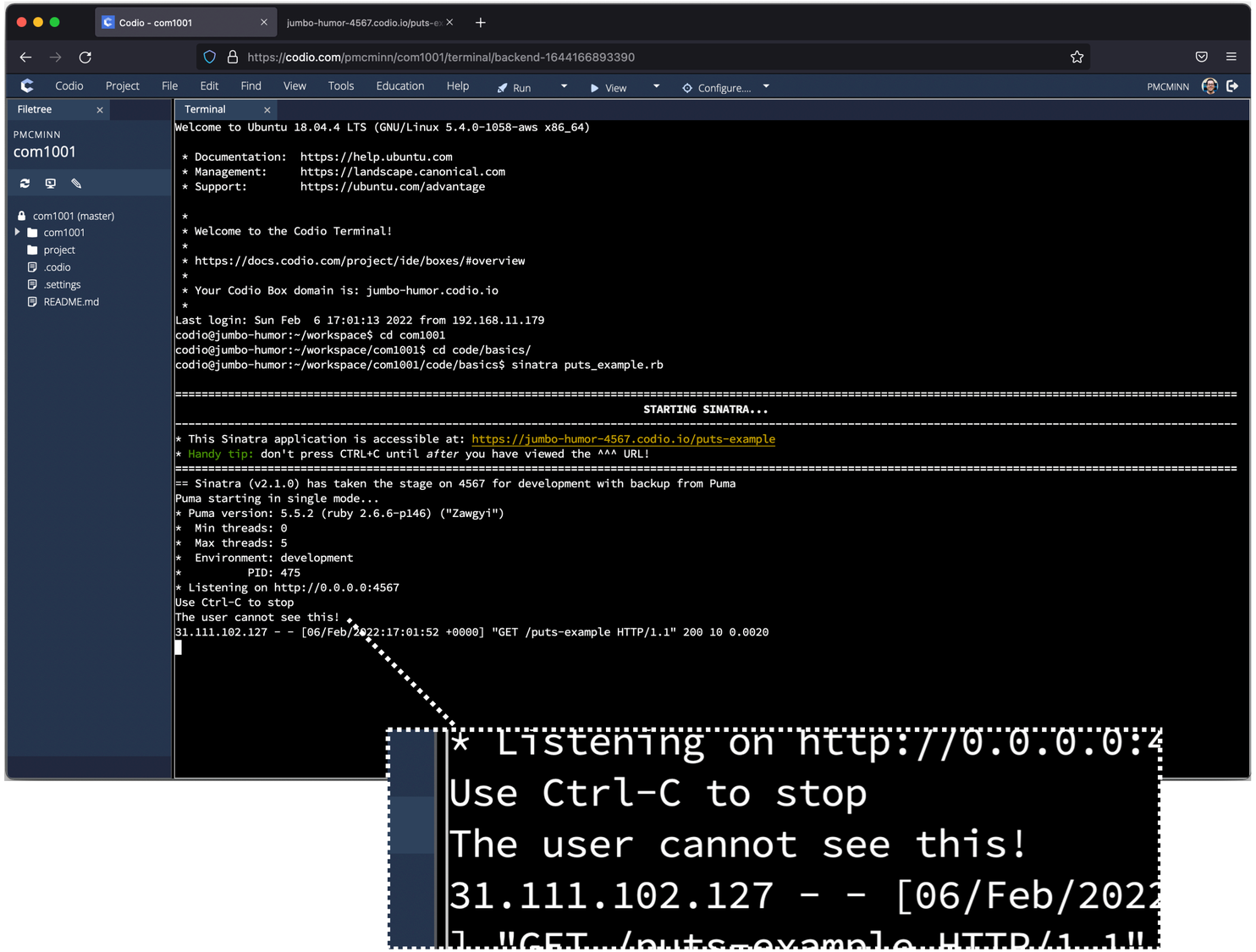
The puts method can still be used – it outputs strings to the console.

The user of the web page cannot see this output of course, as this output exists on the server.

It is therefore useful for **debugging** and other **status messages**.

```ruby
require "sinatra"

get "/puts-example" do
  puts "The user cannot see this!"
  "Hi, folks!"
end
```

basics/puts_example.rb



HTTP Methods

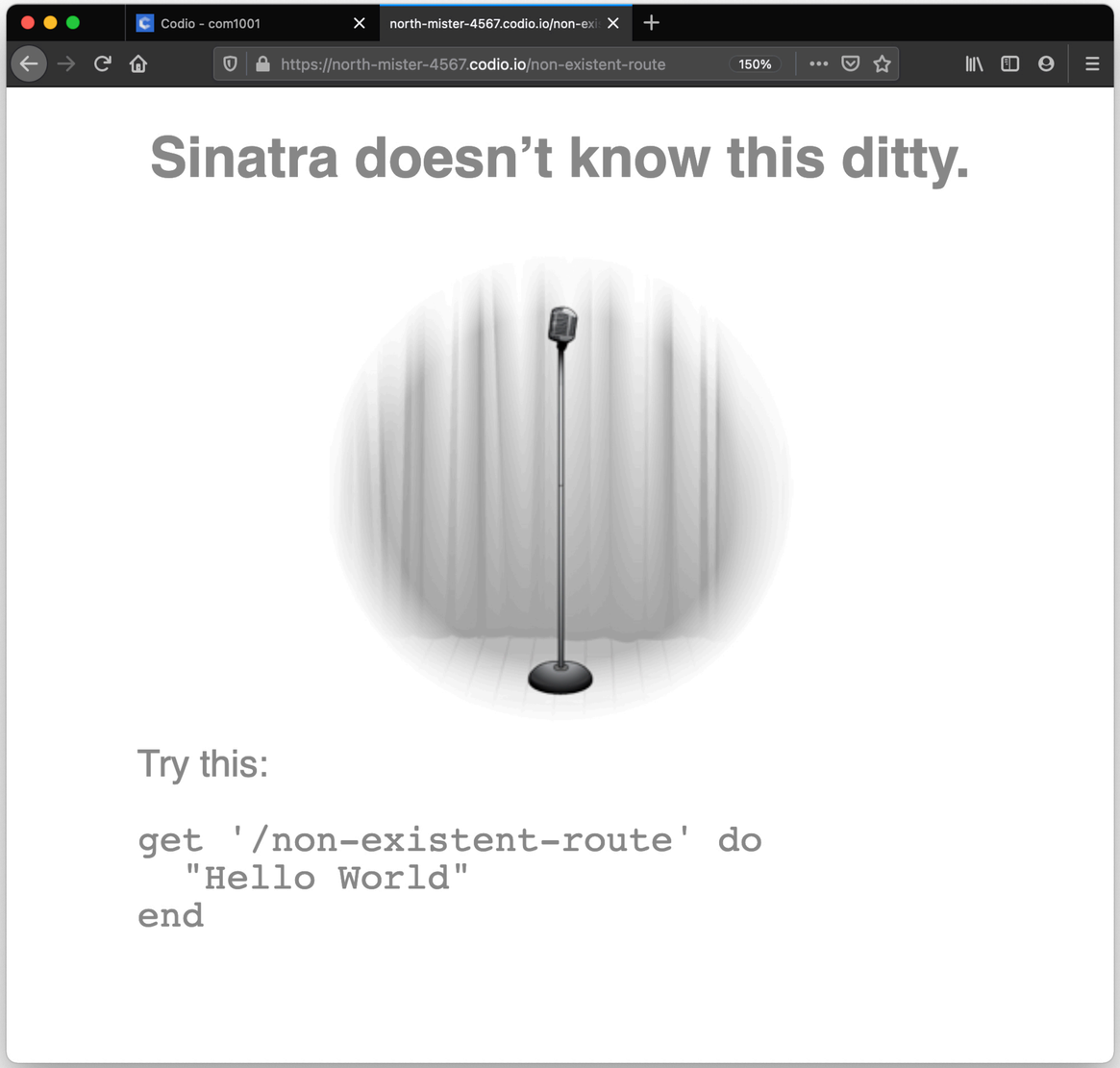
HTTP methods prefix the route. We will only be using the `get` and `post` verbs with Sinatra.

```
require "sinatra"

get "/get-route" do
  "This route is accessed using the HTTP GET method"
end

post "/post-route" do
  "This route is accessed using the HTTP POST method. " \
    "It is inaccessible by typing the URL into your browser."
end
```

basics/http_methods.rb

We can only access `get` routes by typing URLs into our browser. We will discuss how to use `post` in later in the module.

When a HTTP Request Doesn’t Map to a Route



If Sinatra cannot map the HTTP request to a block with a matching route, it triggers a `HTTP 404 Not Found` status and returns this page.

This might be due to a problem with our Ruby code, or it could be that a user mistyped the URL into their browser.

Either way, we would want them to see something more user friendly that does not reveal underlying development details.
We will learn how to do this later in the module.

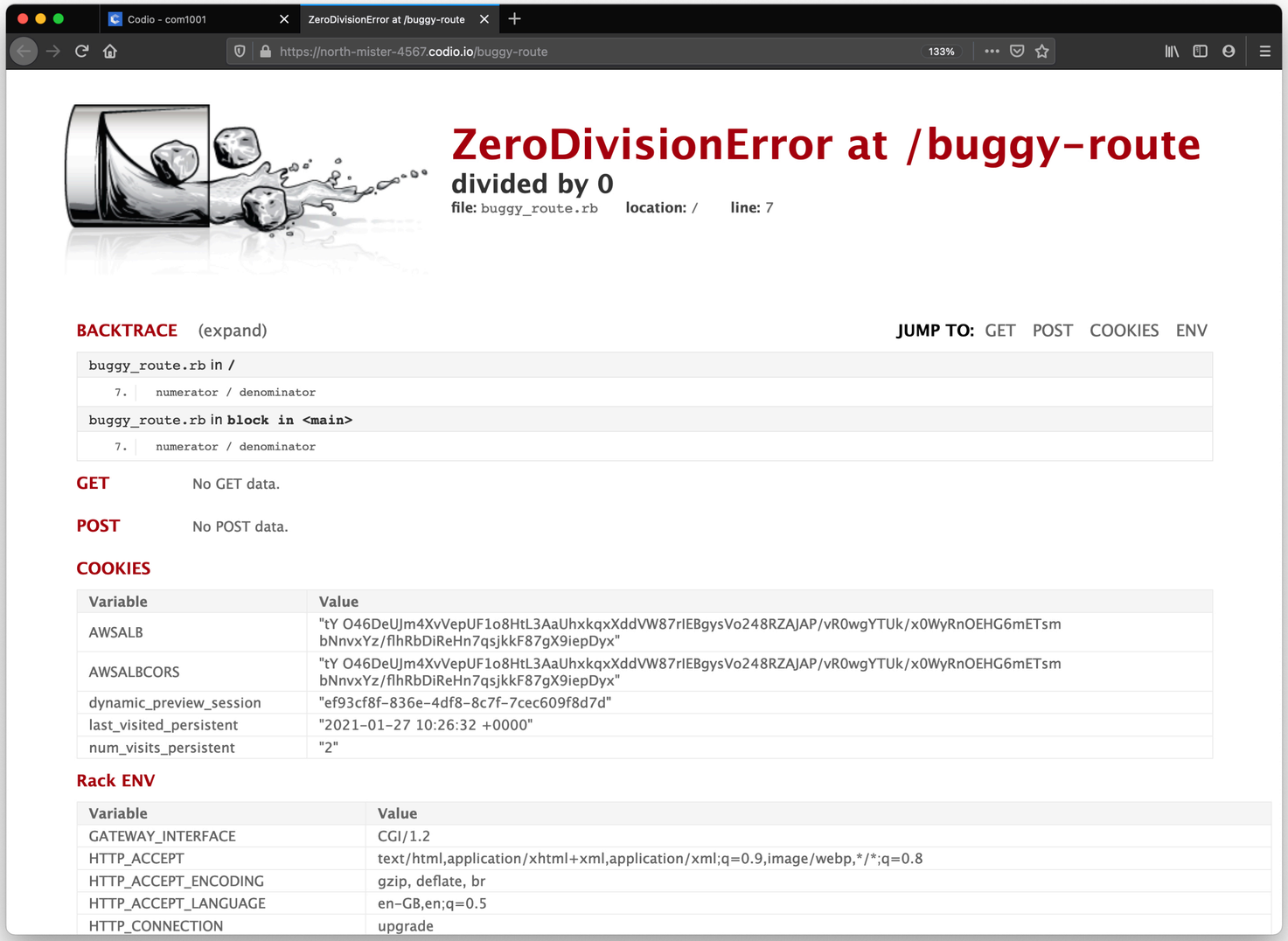
When the Application Contains a Fatal Bug

```ruby
require "sinatra"
set :bind, "0.0.0.0"

get "/buggy-route" do
  numerator = 5
  denominator = 0
  numerator / denominator
end
```

basics/buggy_route.rb

When this route is executed, a **ZeroDivisionError** will occur

When the Application Contains a Fatal Bug



Sinatra encounters an error and so triggers a `HTTP 500 Internal Server Error` status message. In this instance, Sinatra usefully presents us with diagnostic information that may help with debugging.

Again, we would not want the user to see this kind of information once we've deployed the app. We'll return to this later in the module.

Stopping and Starting…

When you change your code, the changes won't take effect until you stop and restart the web server.

This is a bit of pain during the course of normal development. Using `sinatra/reloader` means you don't have to stop and restart the web server after every change. It listens for changes and reloads the application on the fly.

All you need to do is include this line of code in your app, and ensure the `sinatra-contrib` gem is installed.

```ruby
require "sinatra"
require "sinatra/reloader"

get "/reloader-example" do
  "Change me, save, and reload the web page"
end
```

basics/reloader-example.rb

```ruby
source "https://rubygems.org" do
  # ...
  gem "sinatra"
  gem "sinatra-contrib"
end
```

basics/Gemfile