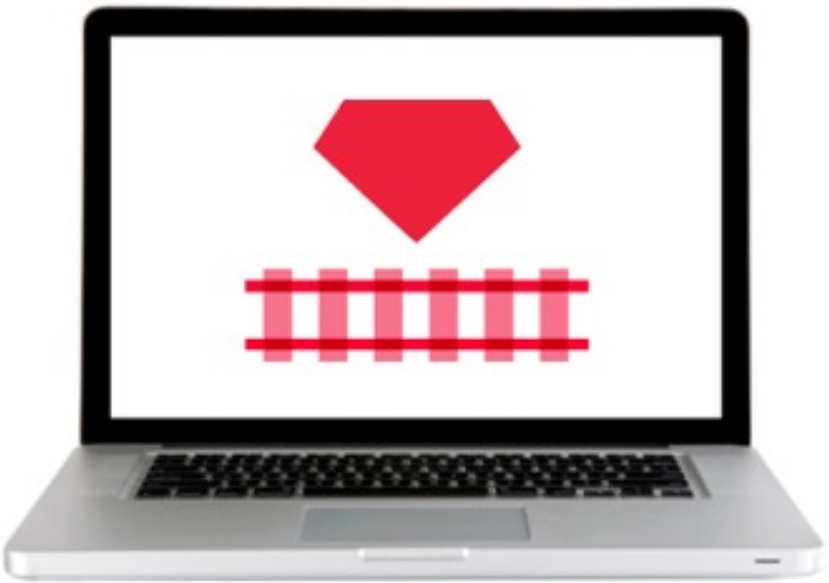


GENERAL ASSEMBLY

# RUBY ON RAILS BASICS

Vincent Trivett

# AGENDA

- Ruby and the Command Line
- HTML — Building blocks
- MVC for web apps
- Creating our first web app
- Deploying our first app to Heroku

# INSTALLATION AND SETUP

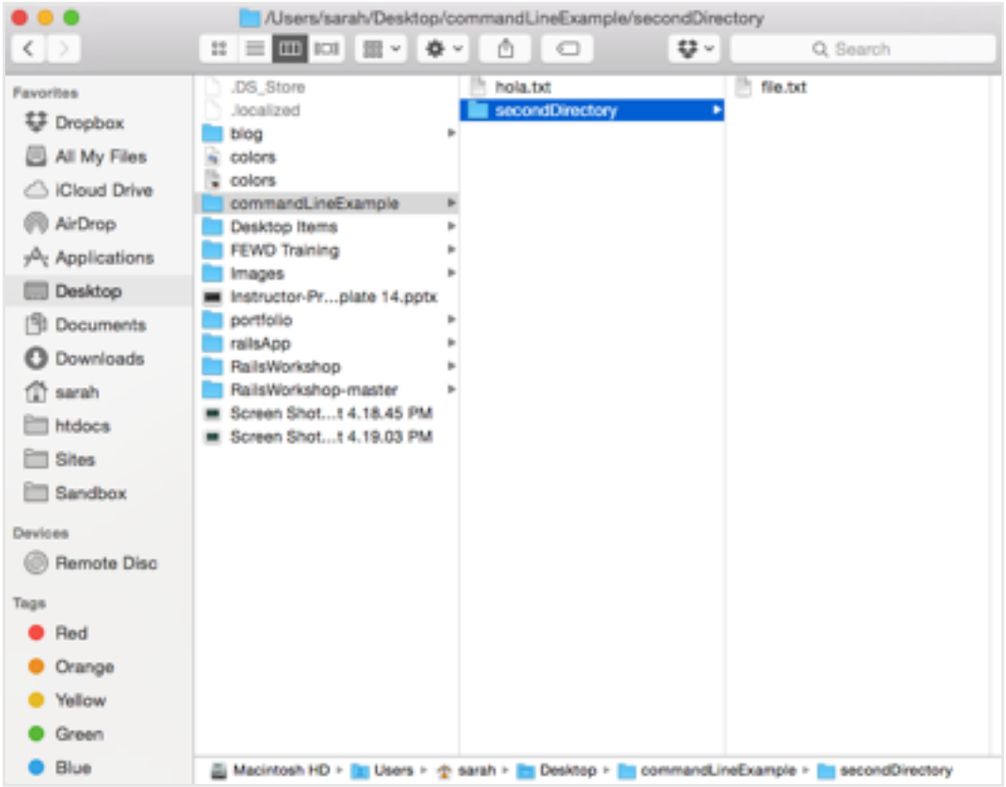
# DO YOU HAVE THE FOLLOWING INSTALLED?

- Ruby
- Rails
- Sublime Text
- Git

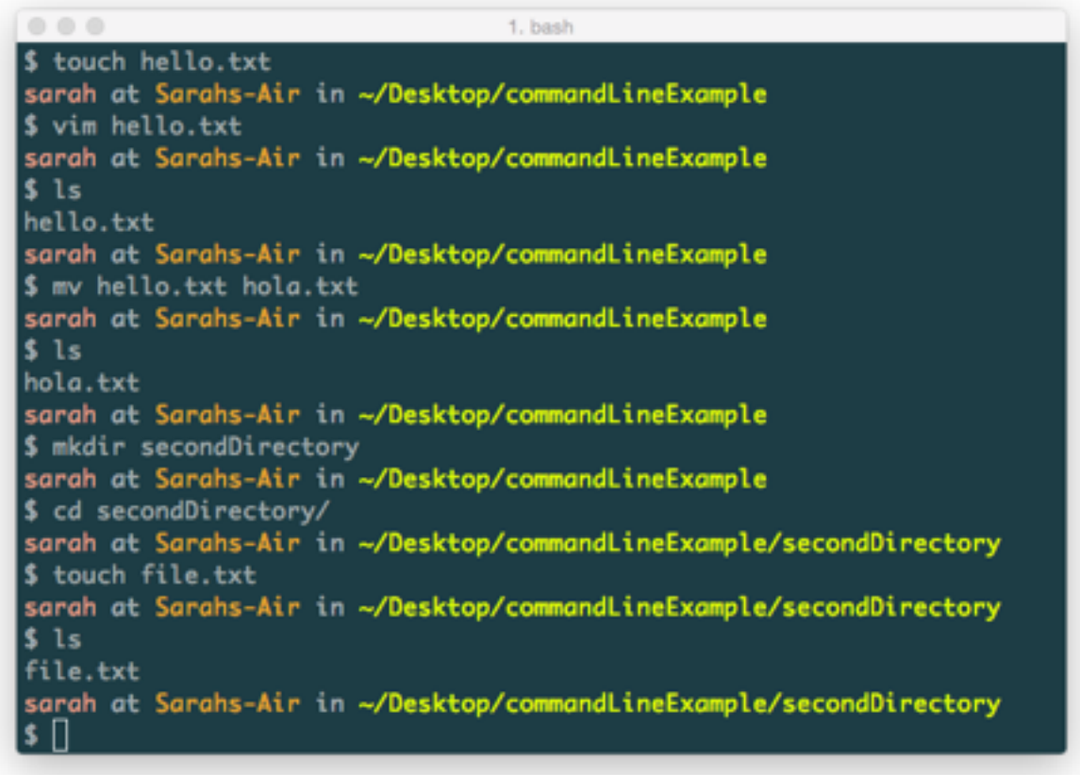
# RUBY AND THE COMMAND LINE

# THE COMMAND LINE

## Non-Developer Workflow




## Developer Workflow



# WHAT CAN I DO WITH THE COMMAND LINE?

## ADD FOLDER

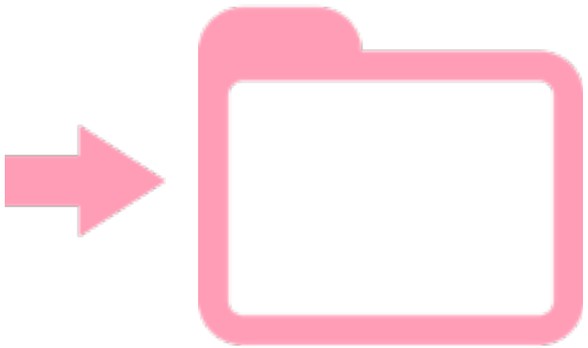
## REMOVE FOLDER

## LIST CONTENTS

## ADD FILE

## REMOVE FILE

## NAVIGATE TO ANOTHER FOLDER



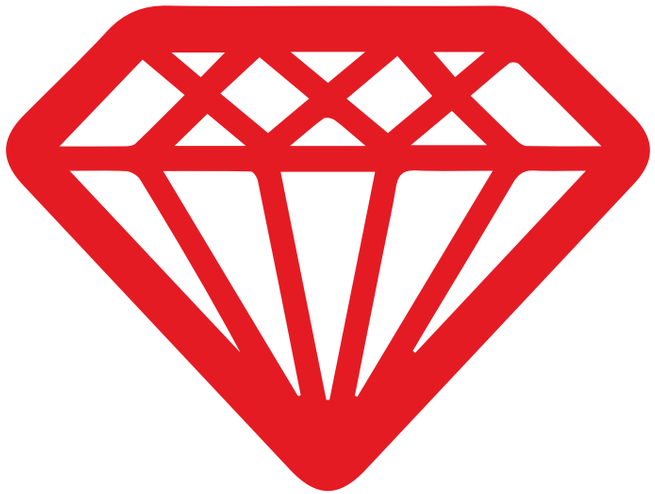
# COMMAND LINE — CODE ALONG!

# RUBY ON RAILS

- **Ruby is a programming language**
- Focus on simplicity and productivity.
- It has an elegant syntax that is natural to read and easy to write.

- **Rails is a framework written in the Ruby language**
- Open source web application framework that runs on Ruby
- Allows you to create web applications that query a database.
- Makes assumptions about what every developer needs when starting a project
- Write less, accomplish more

# IT WILL BE EASIER TO UNDERSTAND RAILS IF YOU UNDERSTAND RUBY FIRST



# AN ELEGANT AND ARTFUL LANGUAGE

- Created by Yukihiro “Matz” Matsumoto - who often said that he was “trying to make Ruby natural, not simple,” in a way that mirrors life.
- It looks and reads a lot like regular English
- Ruby is an **object-oriented** language — *everything* in Ruby is an object

*“Ruby is designed to make programmers happy”*

# IRB

# ARITHMETIC OPERATORS

| Arithmetic Operators | | |
|---|---|---|
| Description | Operator | Example (a = 4 and b = 2) |
| Addition | `+` | `1 + 1` |
| Subtraction | `-` | `3 - 2` |
| Multiplication | `*` | `5 * 3` |
| Exponent | `**` | `3 ** 2 ("3 to the power of 2")` |
| Division | `/` | `10 / 2` |
| Modulus (returns remainder) | `%` | `10 % 3 (would equal 1)` |

## VARIABLES

- Variables store values.
- A value can be assigned to a variable using the = operator.
- Typically start with a lowercase letter and use _ to separate words.
- Ruby is *case-sensitive*, meaning capitalization counts!
- Variables starting with $ and @ have mean different things in Ruby, so best to just start with a lowercase letter

```
variable_name = value
```

# STRINGS

- Created using the double quote character “ ”
- Strings can be added to each other “hello” + “ world” = “hello world”

```
“ ”                # An empty string
“Hello world”      # A non-empty string
```

# STRINGS — INTERPOLATION

- Another way to build up strings is via **interpolation** using the syntax #{}

```
first_name = "Sarah"
"#{first_name} Holden"   => "Sarah Holden"
```

## NUMBERS

1 2 3 2.34 3.14159265359

## BOOLEANS

true false

## COMMENTS

- Start with a pound sign (#) and extend to the end of the line
- Ruby ignores
- Useful to future readers (including author!)

```
# This is a comment
```

```
> 1 + 2   # Here is some basic addition

=> 3
```

# PRINTING AND GETTING USER VALUES

## PRINT

Takes whatever you give it and prints it to the screen

```
print “Hello”
```

## PUTS

- Similar to print, but adds a blank line after it prints your value
- Stands for “put ess” - or “put string”

```
puts “Hello”
```

## GETS

Gets a value from the user (which can be saved to a variable)

```
response = gets
```

# STRINGS — SMASHABLE

# LOGICAL OPERATORS

| Description | Operator | Example (a = 4 and b = 2) |
|---|---|---|
| Equal | `==` | `a == b` *false* |
| Not equal | `!=` | `a != b` *true* |
| Greater than | `>` | `a > b` *true* |
| Less than | `<` | `a < b` *false* |
| Greater than or equal to | `>=` | `a >= b` *true* |
| Less than or equal to | `<=` | `a <= b` *false* |
| Same value and data type? | `.eql?` | `1.eql?(1.0)` *false* |

# BOOLEANS — SMASHABLE

# DECISION TIME!

- Either TRUE or FALSE (like booleans)
- Allows us to select different outcomes depending on user input or the result of a computation
- Called **control flow**

```
if [condition]
  do something
end
```

```
if 1 > 2
  puts “greater than”
end
```

# CONDITIONAL LOGIC – ELSE, ELSIF

```
if [condition]
  do something
elsif [condition]
  do something
else
  do something
end
```

```
if 1 > 2
  puts “greater than”
elsif 1 == 2
  puts “equal”
else
  puts “less than”
end
```

# THE WHILE LOOP

- Will execute a block of code **as long as** the condition is true
- When the condition becomes **false**, the code after the end of the loop will be executed

```
while condition_is_true
  # do something
end
```

# THE UNTIL LOOP

- Will execute a block of code **until** a condition is true
- Similar to while, except backwards
- When the condition becomes **true**, the code after the end of the loop will be executed

```
until condition_is_true
  # do something
end
```

# STORING LISTS OF VALUES

- An array is a data type that holds an ordered collection of values
- Can hold any be any type of object, numbers, strings, even other arrays!
- An array can be used to store a list of values in a single variable

There are 2 different ways to create an array:

```
variable = Array.new
```

```
variable = []
```

```
shapes = [“circle”, “triangle”, “square”]
```

# ACCESSING ITEMS BY INDEX

- Each item in an array has an **index**, by which you can access that item.
- The first item has an index of 0, the second item 1, the third item 2, etc.

```
array = [5, 3, 2, 1, 345]
```

0 1 2 3 4

- Numbers are **zero offset** — which means they start with 0.

To get the second item:

```
array[1]
```

To add an item:

```
array << 12
```

# KEY-VALUE PAIRS

- Collections of **key-value** pairs
- Have indexes, similar to arrays, but their indexes are called **keys**
- Indices don't have to be numbers

```
hash = {}    #defining an empty hash
```

```
hash = {     #defining a hash with key/value pairs
  key1: value1,
  key2: value2
}
```

**Example:**

```
user = {
  name: "Sarah",
  age: 29
}
```

## SYMBOL

- A name used in your program
- Will only keep one copy of a symbol in memory at a time - saves space

```
:symbol
```

## TWO WAYS OF SAYING THE SAME THING

**Hash Rocket:**

```
my_hash = {:key1 => value, :key2 => value}
```

**Key followed by colon and value:**

```
my_hash = {key1: value, key2: value}
```

## MANIPULATING HASHES

- Outside of defining a hash, you need to use the :symbol syntax to denote a symbol.

### ACCESS HASH VALUE

```
my_hash[:key3]
```

### ADDING AN ITEM TO A HASH

```
my_hash[:key3] = value
```

# EACH

## ITERATE OVER AN ARRAY

- Does something for each item in the array
- Takes a variable between | |, which is a placeholder for the item of the array you are currently on.

```
array = [1, 2, 3, 4, 5]
```

```
array.each do |item|
  do something
end
```

## ITERATE OVER A HASH

- Similar to a iterating over arrays
- Needs two placeholder variables to represent each key/value pair

```
hash = [key1: val1, key2: val2]
```

```
hash.each do |key, value|
  do something
end
```

# KEEP YOUR CODE "DRY"

- Groups program logic together so you don't have to repeat yourself
- Can pass variables to methods

**Define:**

```
def method_name(arguments)
  # Code to be executed
end
```

**Call:**

```
method_name(arguments)
```

# BLOCKS

- Blocks are similar to methods, but aren't named
- Similar to 'anonymous functions' in Javascript
- Can be defined with the keywords **do** and **end** or with **curly braces**

```
[1, 2, 3, 4].each do |item|
  # do something
  # and something else
end
```

- **do** and **end** syntax
- Used when blocks are more than one line

```
[1, 2, 3, 4].each { |item| puts item}
```

- **Curly brace** syntax
- For one-line blocks

## METHODS VS. BLOCKS

- Blocks are similar to methods, but aren't named.

**Block:**

```
def method_name(arguments)
  # Code to be executed
end

method_name(arguments)
```

- Can be called infinitely

**Method:**

```
[1, 2, 3, 4].each { |item| puts item}
```

- Can only be called once

## RETURN VALUES

- Sometimes we don't just want a method to do something, we want it to hand us back a value.
- That's where **return** comes in handy.

```
def method_name(arguments)
  return value
end
```

# METHODS — SMASHABLE

# CLASSES

- By convention, class names start with a capital letter and use CamelCase instead of underscores
- A class has the ability to create other objects that are of its kind.

```
class myClass
  def initialize(param1)
    @param1 = param1
  end
end
```

```
class user
  def initialize(name, age)
    @name = name
    @age = age
  end
end
```

```
newObject = myClass.new(param)
```

```
sarah = user.new("Sarah", 27)
```

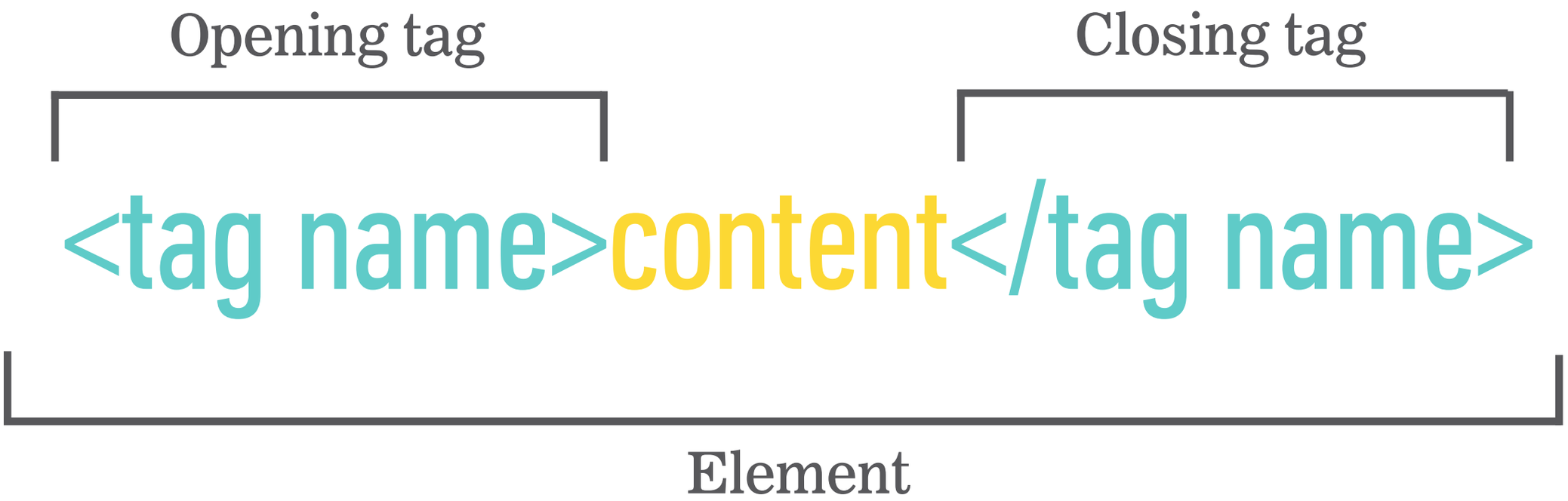
# HTML: BUILDING BLOCKS

## WHAT IS HTML?

HTML describes the organization and structure of pages

## WHAT IS HTML?

HTML describes the organization and structure of pages



# MVC FOR WEB APPS

# WHAT IS MVC?

*An Architectural pattern that describes a way to structure our application and the responsibilities and interactions for each part in that structure.*

# MVC – LET'S BREAK IT DOWN

**M** MODEL

**V** VIEW

**C** CONTROLLER

# MVC

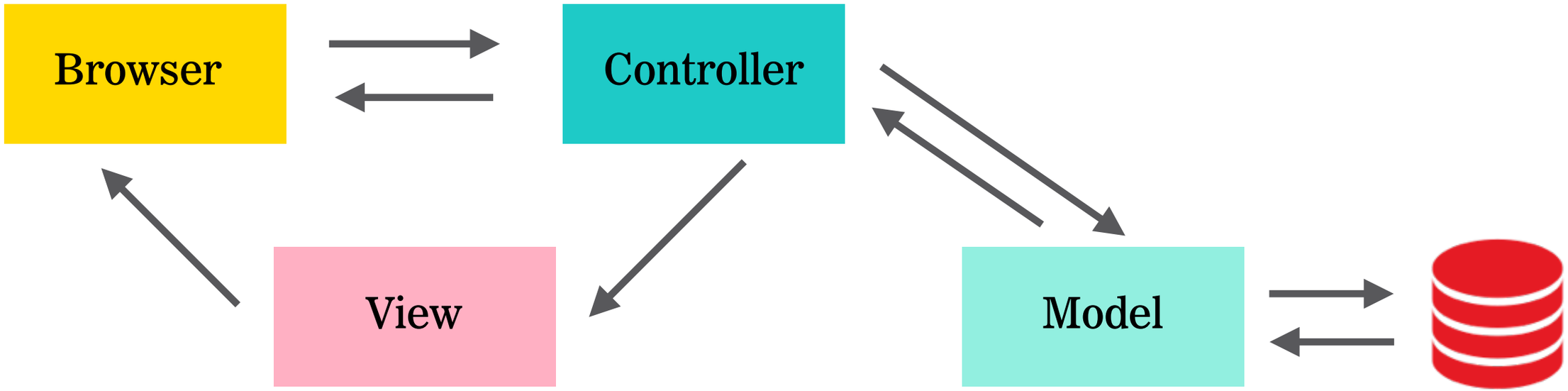
**M** MODEL — DATA

**V** VIEW — PRESENTATION

**C** CONTROLLER — DECISIONS

# BASIC WEB ARCHITECTURE

# MVC



# CREATING OUR FIRST WEB APP

## DRY

- Don’t repeat yourself!!!!
- Consise, consistent code that is easy to maintain

## CONVENTION OVER CONFIGURATION

- Sensible defaults.
- Makes assumptions about what developers need to start a project
- Speeds up development, there’s not as much code to maintain

# HELLO WORLD!

# GENERATE A RAILS APPLICATION

- First step with any new application/project

```
$ rails new projectName
```

## RAILS SERVER

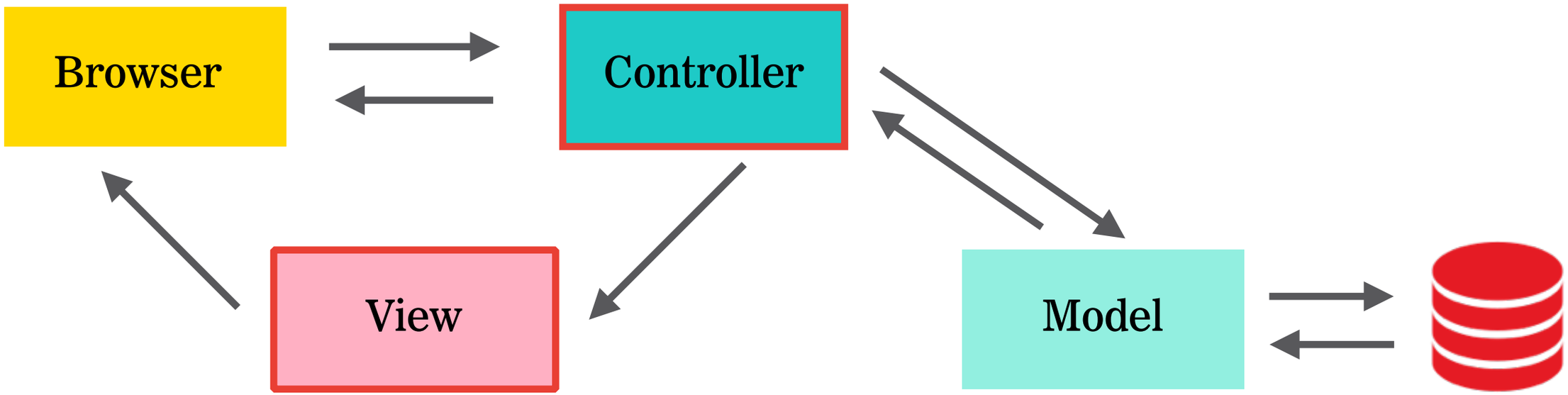
- Launches a small web server that comes with ruby.
- Use this command to access the webpage within your browser

```
$ rails server
```

**Shorthand:**

```
$ rails s
```

# MVC



## RAILS GENERATE

- Uses templates to create a whole lot of things
- Save time by writing **boilerplate code** — code that's necessary for the app to work

```
$ rails generate
```

**Shorthand:**

```
$ rails g
```

## GENERATE CONTROLLER

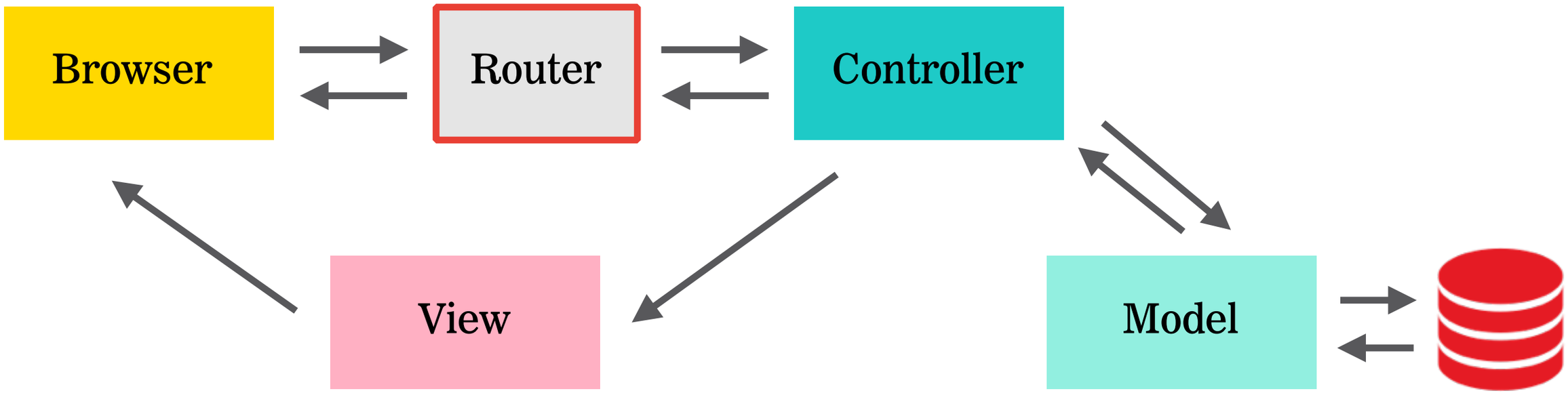
- Controller's purpose is to receive specific requests for the application

```
$ rails generate controller name action
```

```
$ rails generate controller welcome index
```

# MVC — ROUTER

- **Routing** decides which controller receives which requests.



# GIT

# INITIALIZE A REPOSITORY

- Repository - Place where the history of your work is stored
- First command you'll run in a new project
- Creates .git subdirectory in the project root
- Only needs to be executed once per project

```
$ git init
```

## GIT ADD

- Adds a change in the working directory to the staging area.
- Tells git that you want to include those changes in the next commit

## GIT COMMIT

- Saves a copy or ‘snapshot’ of the current state of the project.
- These are ‘safe’ versions of the project - you can go back to them and git won’t do anything to them unless you tell it to.

```
$ git add
```

```
$ git commit
```

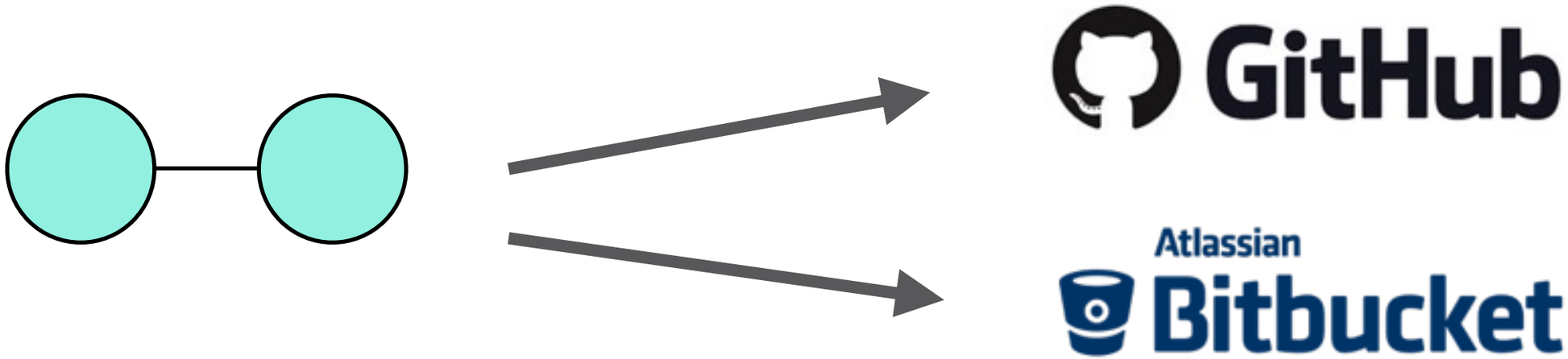
## GIT STATUS

- Lists which files are staged, unstaged, and untracked.

```
$ git status
```

# GIT PUSH

- Transfer local commits to remote repository



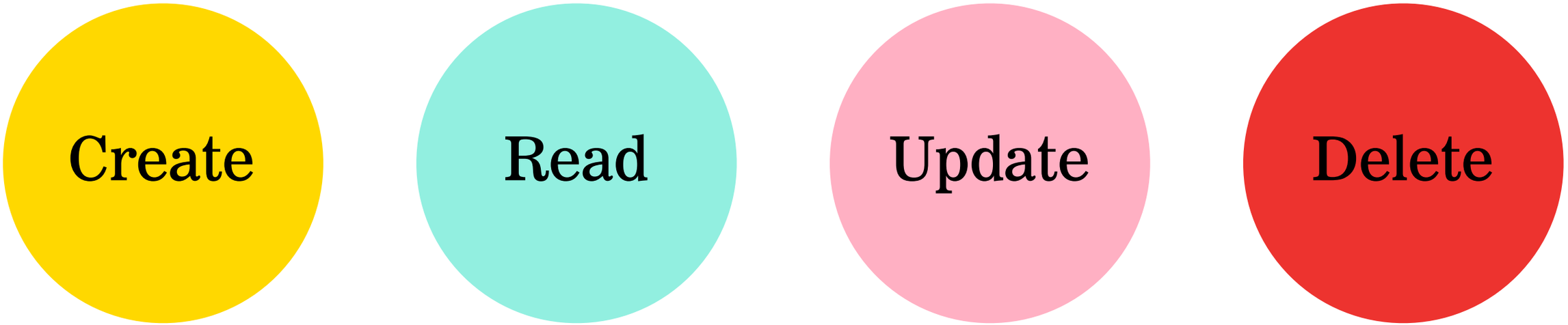

```
$ git push
```

# CREATING OUR FIRST WEB APP — PART II

# SOMETHING WITH A BIT MORE SUBSTANCE — A BLOG

## RESOURCES

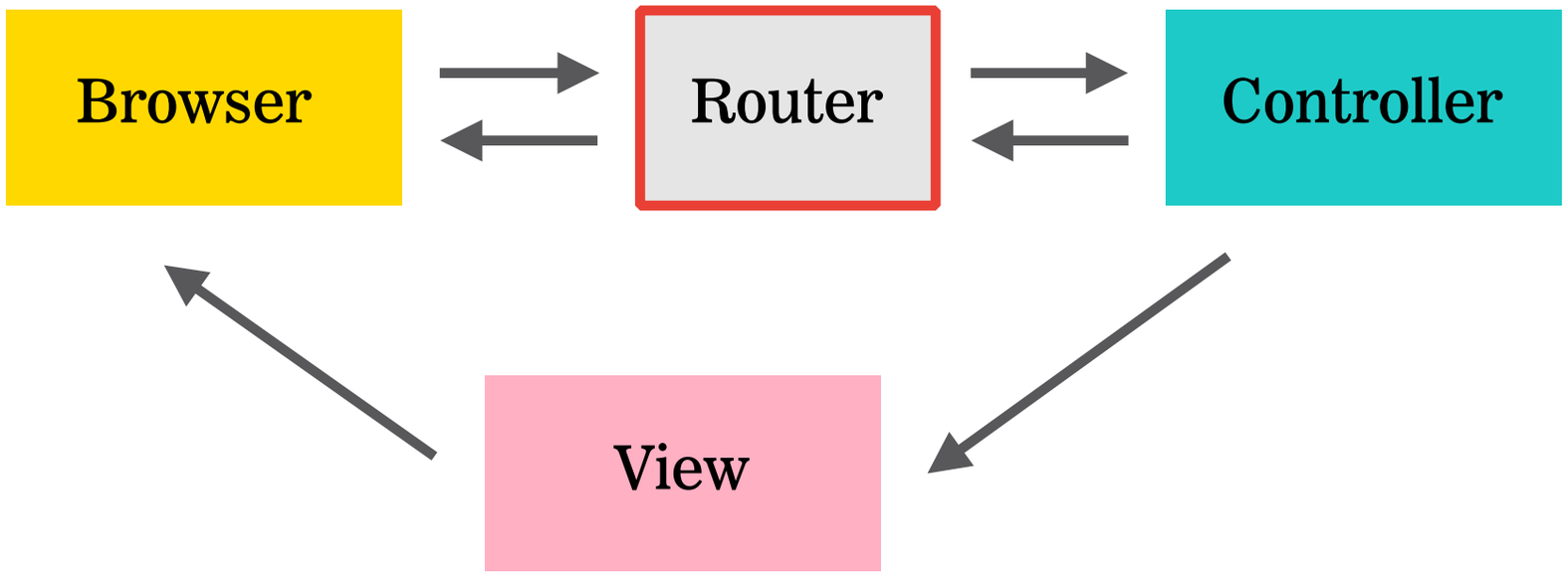
- A **Resource** is a term used for a collection of similar objects, such as articles, people, or animals.
- You can create, read, update and destroy items for a resource - called **CRUD**




```
resources :users          # declared in routes.rb file
```

# WE'VE ADDED A ROUTER. WHAT'S NEXT?

# WE'VE ADDED A CONTROLLER. WHAT'S NEXT?



# ERB

**Embedded Ruby** - A simple template language

%= Replace ruby code with actual value

```
<%= ruby code %>
```

% Perform an operation in Ruby

```
<% ruby code %>
```

## OUR FIRST FORM

- Use the form builder to create a form.
- Primary form builder in Rails uses a helper method called **form_for**

```
<%= form_for :object, url: object_path do |f| %>
   # form goes here (inputs, labels, submit button)
<% end %>
```

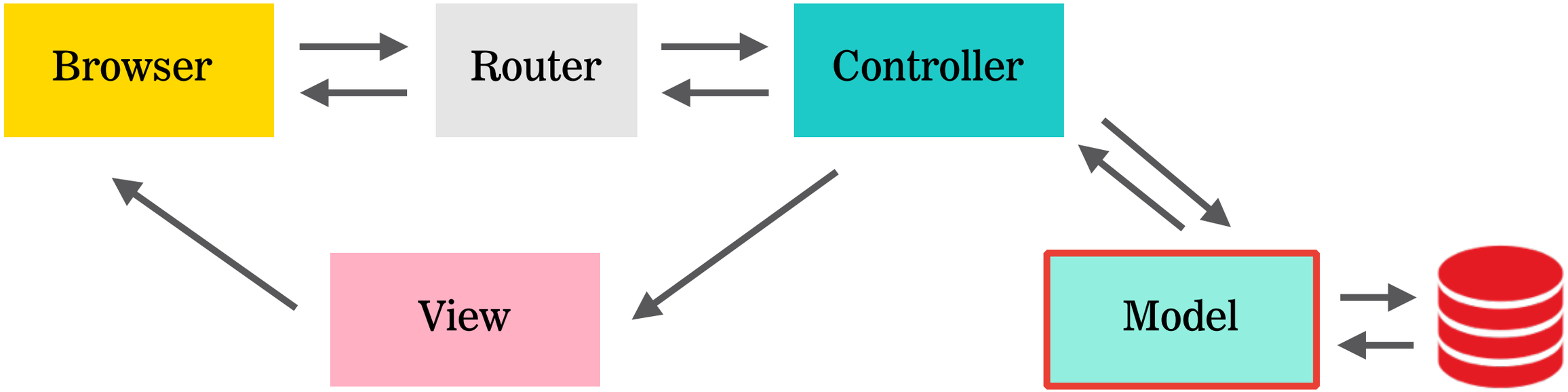
```
<%= form_for :article, url: articles_path do |f| %>
   # form goes here (inputs, labels, submit button)
<% end %>
```

# ERB & HTML OUTPUT

```
<%= form_for :article do |f| %>
    <%= f.label :title %>
    <%= f.text_field :title %>

    <%= f.label :text %>
    <%= f.text_area :text %>

    <%= f.submit %>
<% end %>
```

```
<form action="/articles/new" method="post">
    <label>Title</label>
    <input type="text">

    <label>Text</label>
    <textarea></textarea>

    <input type="submit">
</form>
```

# COMMUNICATING WITH THE DATABASE



## GENERATING A MODEL

- Models in rails use a **singular** name
- Rails provides a generator for creating models

```
$ rails generate model NAME [field[:type][:index] field[:type][:index]]
```

```
$ rails generate model article title:string text:text
```

## DATABASE MIGRATIONS

- When we generate a model, it creates a migration file in our db directory
- **Migrations** are Ruby classes that make it simple to create and modify database tables

```
class CreateArticles < ActiveRecord::Migration
  def change
    create_table :articles do |t|
      t.string :title
      t.text :text

      t.timestamps
    end
  end
end
```

| articles | |
|---|---|
| id | integer |
| title | string |
| text | text |

## RUNNING A MIGRATION

- Rails uses **rake** commands to run migrations

```
$ rake db:migrate
```

# CREATE

Create

Read

Update

Delete

# SAVING DATA IN THE CONTROLLER

Create

```
def create
  @article = Article.new(params[:article])

  @article.save
  redirect_to @article
end
```

## STRONG PARAMETERS

Create

- Rails has several security features that help you write secure applications
- Rails uses **strong_parameters**, which requires us to tell Rails exactly which parameters we want to accept in our controllers

```
def create
  @article = Article.new(article_params)

  ...
end

private
  def article_params
    params.require(:article).permit(:title, :text)
  end
```

# READ

Create

Read

Update

Delete

## SHOWING AN ARTICLE

Read

To show an article, we need a **show** action in our controller

```
def show
    @article = Article.find(params[:id])
end
```

**@article** is an instance variable, it allows us to access our article from the view

## LISTING ALL ARTICLES

Read

To show a list of articles, we need an **index** action in our controller

```
def index
    @articles = Article.all
end
```

# ADDING LINKS

The **link_to** helper creates an <a> or ‘anchor’ tag with the given text and URL

```
<%= link_to “Text to display”, url %>
```

```
<%= link_to “Show”, contact %>
```

# UPDATE

Create

Read

Update

Delete

# UPDATING AN ARTICLE – THE EDIT PAGE

Update

We need an ‘edit’ action in our controller for our edit page

```
def edit
  @article = Article.find(params[:id])
end
```

## UPDATING AN ARTICLE – SAVING CHANGES

Update

We'll need an **update** action in our controller to save our changes
Remember, we'll also need **strong parameters**

```
def update
  @article = Article.find(params[:id])

  if @article.update(article_params)
    redirect_to @article
  else
    render 'edit'
  end
end

private
  def article_params
    params.require(:article).permit(:title, :text)
  end
```

# DELETE

Create

Read

Update

Delete

# DELETING ITEMS

Delete

- We use the **delete** method for destroying resources, and this route is mapped to the **destroy** action
- Different from a typical link so that someone can't maliciously delete something from the database via a link

```
<a href="href='…/articles/1/destroy'>Tricky link!</a>
```

```
<%= link_to 'Destroy', item_path(item),
                    method: :delete, data: { confirm: 'Are you sure?' } %>
```

## WE HAVE A WEB APP!

Congrats, we did it!

Pat yourself on the back!!!

# PARTIALS

- A view that’s included as part of another view
- File name is preceded by an underscore

- Example: _form.html.erb

```
<%= render ‘form’ %>
```

# DEPLOYING TO HEROKU

# NEXT STEPS

# GENERAL ASSEMBLY — FULL-TIME COURSES

Filter by topic | ALL FORMATS | FULL-TIME COURSES | PART-TIME COURSES | CLASSES & WORKSHOPS | EVENTS

Make a career change with our 8–12 week, all-day, full-time programs.

**Mon, 9 March**

WDI

## Web Development Immersive

12-weeks. All day, every day. Learn the skills to become an entry-level web developer and the resources to get a job in this intensive program.

*12 Week Course*

9 Mon, 9 March
9:00 - 5:30pm

*Info Session:*
Thu, 11 December at 6:00pm

# GENERAL ASSEMBLY — PART-TIME COURSES

Filter by topic | ALL FORMATS | FULL-TIME COURSES | **PART-TIME COURSES** | CLASSES & WORKSHOPS | EVENTS

**Mon, 12 January**

FEWD

## Front-End Web Development

In this 10-week course, students learn to code, speak the language and implement their own designs by learning HTML, CSS, and JavaScript.

*10 Week Course*

12 Mon, 12 January
6:00 - 9:00pm

**Tue, 13 January**

UXD

## User Experience Design

In this 12-week course, students learn to build wireframes, implement best practices for common design patterns and analyze business goals from a user perspective.

*12 Week Course*

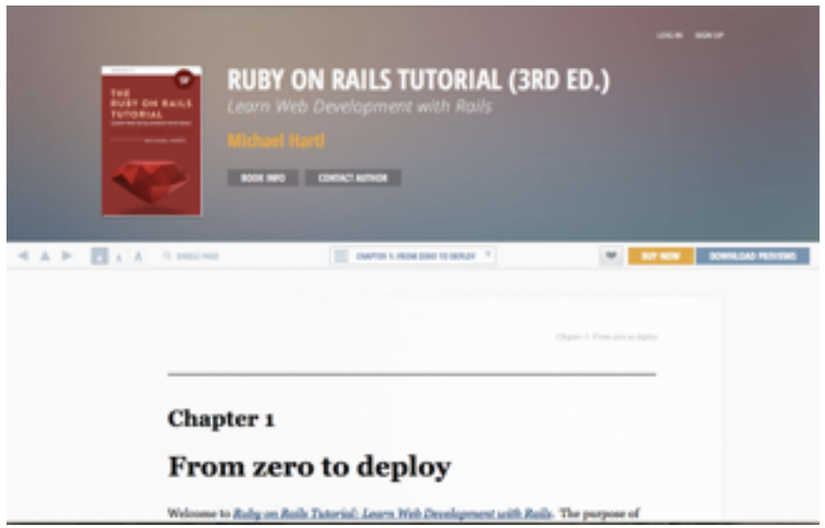
13 Tue, 13 January
6:30 - 8:30pm

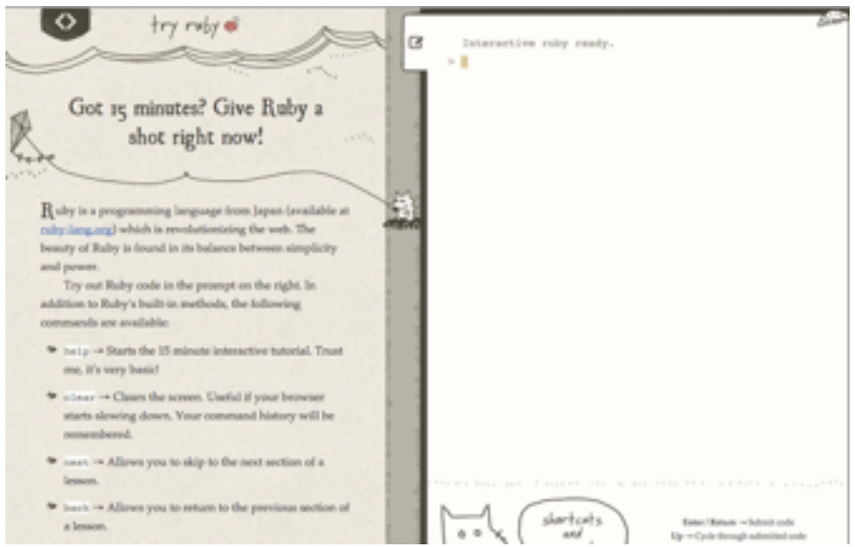
*Info Session:*
Mon, 24 November at 5:30pm

**Mon, 9 March**

# ONLINE LEARNING





# Q&A