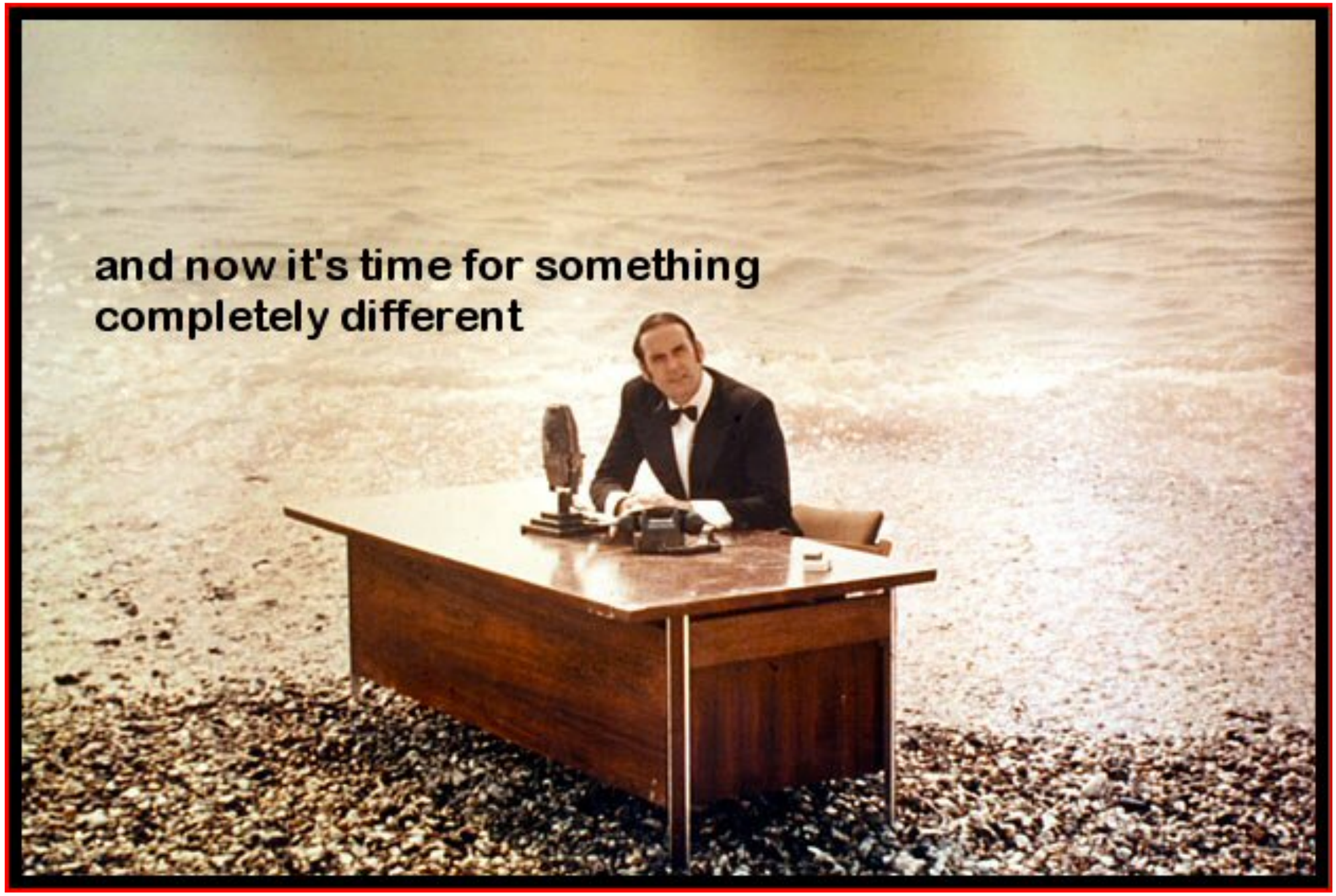
And welcome back

RECAP O'CLOCK

- Remember Ruby?
- As a developer, you will frequently come against stuff that you have zero familiarity with in other people's code.
- Not everyone documents their code well.
- See how well you can figure out this code. Before you start researching the unfamiliar stuff, get as far as you can with what you do know.
- https://gist.github.com/trivett/d629ba54ccd089032b4f
- You got 15 minutes!


and now it's time for something
completely different

FILL OUT THE SURVEY IN YOUR EMAIL

It's rather important

When you finish, start thinking about the various models and gems you will need in your final project.

OBJECTIVES

- You will learn what a form is, and how to accept data from your users.
- You will learn how to store user typed information inside of your Database (using a model)

FORMS

- Forms are used to collect data from users on a webpage
- They commonly map directly to a model's attributes
- A common form you use all the time is "sign up" or "login" "new tweet" etc.

FORMS (CONT)

- Forms contain “fields”, a field maps to a single attribute
- There are many types of fields (password, text, select, textarea, …)
- Rails contains helpers that allow you to create forms very easily

FORMS (CONT)

- When you go to an index or show page, you make one type of HTTP request.
- When you delete a record, you make another.
- What type of request happens when you submit a form?

FORMS (CONT)

- When you submit a form, it performs a different type of request.
- We've defined routes using "get …", but forms will use the method "post" and "put"
- So you will have routes that are similar to "post '/create'" now

RESOURCE ROUTES

- `resources :something`
- Defines 7 routes in 1 line of code
- The routes that are defined are the most common routes you will used.

routes.rb

```ruby
Rails.application.routes.draw do
  resources :photos
end
```

Creates these routes:

localhost:3000/rails/info/routes

Routes

Routes match in priority from top to bottom

Helper	HTTP Verb	Path	Controller#Action
Path / Url		Path Match	
photos_path	GET	/photos(.:format)	photos#index
	POST	/photos(.:format)	photos#create
new_photo_path	GET	/photos/new(.:format)	photos#new
edit_photo_path	GET	/photos/:id/edit(.:format)	photos#edit
photo_path	GET	/photos/:id(.:format)	photos#show
	PATCH	/photos/:id(.:format)	photos#update
	PUT	/photos/:id(.:format)	photos#update
	DELETE	/photos/:id(.:format)	photos#destroy

QUICK START

- In your class 11 folder
- Create a new rails app (rails new) called "`amazing_products`"
- Declare your routes with the resources shortcut
- `cd products`
- Generate a model called "Product" with 2 attributes
  - name:string
  - price:integer
- Migrate your database
- When you finish, add at least three products with seeds or the rails controller.

SHOW, INDEX

- Give the products controller an index and show action.
- Link the products on the index page to the appropriate show action with link_to
- We have already done this, nothing new so far. Look back at the slides from the last class for hints.

ADDING A FORM

- Forms are a part of your view
- They are the portal to interacting with a model (most of the time) from a browser.
- Which type of HTTP request will be required for the form?
- How might you check?

CREATING DATA

- All points of “creating” a new record in your database starts at the controller action “new”
- So the page for adding a product would be:
  - Controller: ProductsController
  - Action: new
  - View: app/views/products/new.html.erb

CREATING DATA

- The new action simply *displays* a form to the user. It does **not** actually create the data in the database.
- Not creating anything in the database.
- The create action is responsible for creating the data within our model.

- Look at `rake routes`
- Notice the different HTTP methods for new and create

OUR CONTROLLER

products_controller.rb

```rb
class ProductsController < ApplicationController
  def new
    @product = Product.new
  end
end
```

- *Only initializes an empty model*
- *This means all of our attributes are **nil. No id. No created_at. Nothing.***
- *This is simply to pass our view (and form) an empty model to "fill out"*

OUR VIEW

new.html.erb

```
1  <%# app/views/products/new.html.erb %>
2
3  <%= form_for @product do |form| %>
4    Name: <%= form.text_field :name %>
5    <br>
6    Price: <%= form.text_field :price %>
7    <br>
8    <%= form.submit "Add Product" %>
9  <% end %>
```

Create and display a form for @product (passed from our controller)

Display a text field for the "name" attribute on the Product model

Display a submit button that has the text "Add Product" inside of it

LAB

- Add a “new” action that sets up the product form
- Add a form for a new product that displays fields for the name and price for the new controller and action.
- Include a submit button
- Put a link to the new action on the index view. Use the handy new_product_path path helper.
- Try it out in the browser. What happens when you hit submit?

BREAK

COLLECTING THE DATA

- Again, the new action only displays the form, it does **not actually create the data.**
- To create the data in our model / database. We need to add another action called create.

PRESSING SUBMIT

- The markup that the `form_for` code creates includes HTML <input> tags. Let's have a look at that in the Chrome developer tools.
- Run your server and right click on the form to inspect element. What's all this?
- When you press "submit" on the form, your browser performs a "POST" request to a URL in your rails application.
- The `form_for` helper will make the submit button send data to "`/products`" with a `POST` request type.

THE CREATE ACTION

- Sending POST /products routes to the Products controller and the create action.
- You can see this with rake routes, or /rails/info/routes in your browser

DATA FORMAT

- In your controller actions, you have access to a hash called "`params`" Yep. THOSE params.
- The `params` hash contains all details from the route, and any data passed to the controller (via POST).
- You use the `params` hash to collect data from a form and save it to a database using your model.

DATA FORMAT

- The params hash is also magical. Rails adds methods to "permit" data from being sent to your controller.
- This is used to prevent values being sent that aren't allowed. For example:
  - Changing the availability of a product
  - Changing the users enabled / disabled status
  - Many others

IF YOU SEE THIS PERSON COVERED IN NUMBERS WITH A HOODIE PLEASE REPORT TO THE POLICE. THIS IS WHAT THEY LOOK LIKE. SI VES ALGO, DI ALGO.

DATA FORMAT

- If the controller doesn't specifically permit certain params a hacker can potentially inject his or her own parameters into the form by guessing attribute names.
- For instance, if you have a user model with name and is_admin, the hacker can guess that if there is an is_admin and it is set to true, passes that parameter, guess what.
- All your data belong to us. pwned. whatever the kids on 4chan are saying these days.

STRONG PARAMETERS

products_controller.rb

```ruby
class ProductsController < ApplicationController
  def new
    @product = Product.new
  end

  def create
    puts params.inspect
    # => { "action" => "create", "controller" => "products", "product" => {
    #        "name" => "user input", "price" => "user input" } }
    product_params = params.require(:product).permit(:name, :price)

    puts product_params.inspect
    # => { "name" => "user input", "price" => "user input" }
  end
end
```

ALL RAILS MODELS CAN BE INITIALIZED WITH A HASH

- Remember, you can always initialize a rails model with a hash:
  - `Product.new(“name” => “Beats Studio”, “price” => 400.50)`
- So if params is just a hash…

INITIALIZING MODELS WITH PARAMS

products_controller.rb

```rb
class ProductsController < ApplicationController
  def new
    @product = Product.new
  end

  def create
    product_params = params.require(:product).permit(:name, :price)

    @product = Product.new(product_params)
    @product.save

    render nothing: true
  end
end
```

(skip rendering a view file)

INITIALIZING MODELS WITH PARAMS

products_controller.rb

```
1  class ProductsController < ApplicationController
2    def new
3      @product = Product.new
4    end
5
6    def create
7      product_params = params.require(:product).permit(:name, :price)
8
9      @product = Product.new(product_params)
10     @product.save
11
12     render nothing: true
13   end
14 end
```

(skip rendering a view file)

Why render nothing??

RECAP

- All data submitted through a form is placed in the “params” hash in our controller actions
- We use the params hash to initialize and save models to our database. Hence creating data from our browser!

CODE ALONG

- Accepting user data from our products form to create a product in our database!

BREAK

REDIRECTS AND FLASHES

- Rails provides 2 facilities that are very useful for creating and updating databases.
- Redirects
  - A controller can redirect the user's browser to another url (and another action at that)
- Flashes
  - A controller can set something called a "flash message" that is only displayed on the next page load.

REDIRECTS

- In a controller action, you can use:
  - redirect_to “/hello_world”
  - This make the users browser go to “/hello_world” on the same domain.
    - this assumes that you have something at that route (or in the public folder)

REDIRECTS

- You use redirects commonly when you may not have a view, but still want to perform an action.
- For example, creating something in our database usually doesn't have a view to be rendered. You typically will redirect the user to another page, and inform them there that the item was created.

- So our create action that creates products, wouldn't actually have "`app/views/products/create.html.erb`" That doesn't make sense. We already saw this with the delete method.
- Instead, you would:
  - `redirect_to "/products"`
    - You can use the path helper that rails gives you
    - `redirect_to products_path`
- And display using a flash (coming up next) that the product was created.

FLASHES

- Another hash!
- Flash hash!
- Flashes are displayed only once on a page
- Typically used for displaying quick info messages in a view.
  Such as “Product created successfully!”

FLASHES ARE ANOTHER SPECIAL HASH

- Rails uses flashes like hashes.
- To define a flash message, you can set a key on the flash hash with the value you'd like.
- For example:
  - `flash[:notice] = "Product created!"`

```ruby
def create
  product_params = params.require(:product).permit(:name, :price)

  @product = Product.new(product_params)
  if @product.save
    flash[:notice] = "Product created!"
    redirect_to "/products"
  end
end
```

```ruby
def create
  product_params = params.require(:product).permit(:name, :price)

  @product = Product.new(product_params)
  if @product.save
    flash[:notice] = "Product created!"
    redirect_to "/products"
  end
end
```

This if statement both SAVES the product and returns true if it worked.

DISPLAYING A FLASH

- You can access the same flash hash in the view!
- Now where can I place the flash message where it will be visible on ALL views…..

DISPLAYING A FLASH

- That's right! views/layouts/application.html.erb. Maybe right above the <%= yield %> statement.
- The application layout is rendered on every page. The bread part of the burger.
- This allows you to not have to render the flash message in every single view file you have. A big time save.

DISPLAYING A FLASH

- Using erb, you can render a flash message with:
- `<%= flash[:notice] %>`
- On the next page load, this won't display anything. It is only displayed once.

CODE ALONG

- Making a flash message appear after our products are created.
- In our controller, make a flash message appear on the next page when a product is created successfully.
- Redirect the user to the products index page

EXIT TICKETS

- Remember to fill out exit tickets

HOMEWORK / LAB

- More work on the Movie app
- Spec
  - Resourceful routes. Set the root to the index route.
  - Index page displaying all movies. Each movie name should be a link to the show page for the movie.
  - Link to delete a movie on that movie's show page.
  - Link to new movie with a form.
  - When a movie is created, redirect to the index page with a message that it worked.
  - Header in application layout.