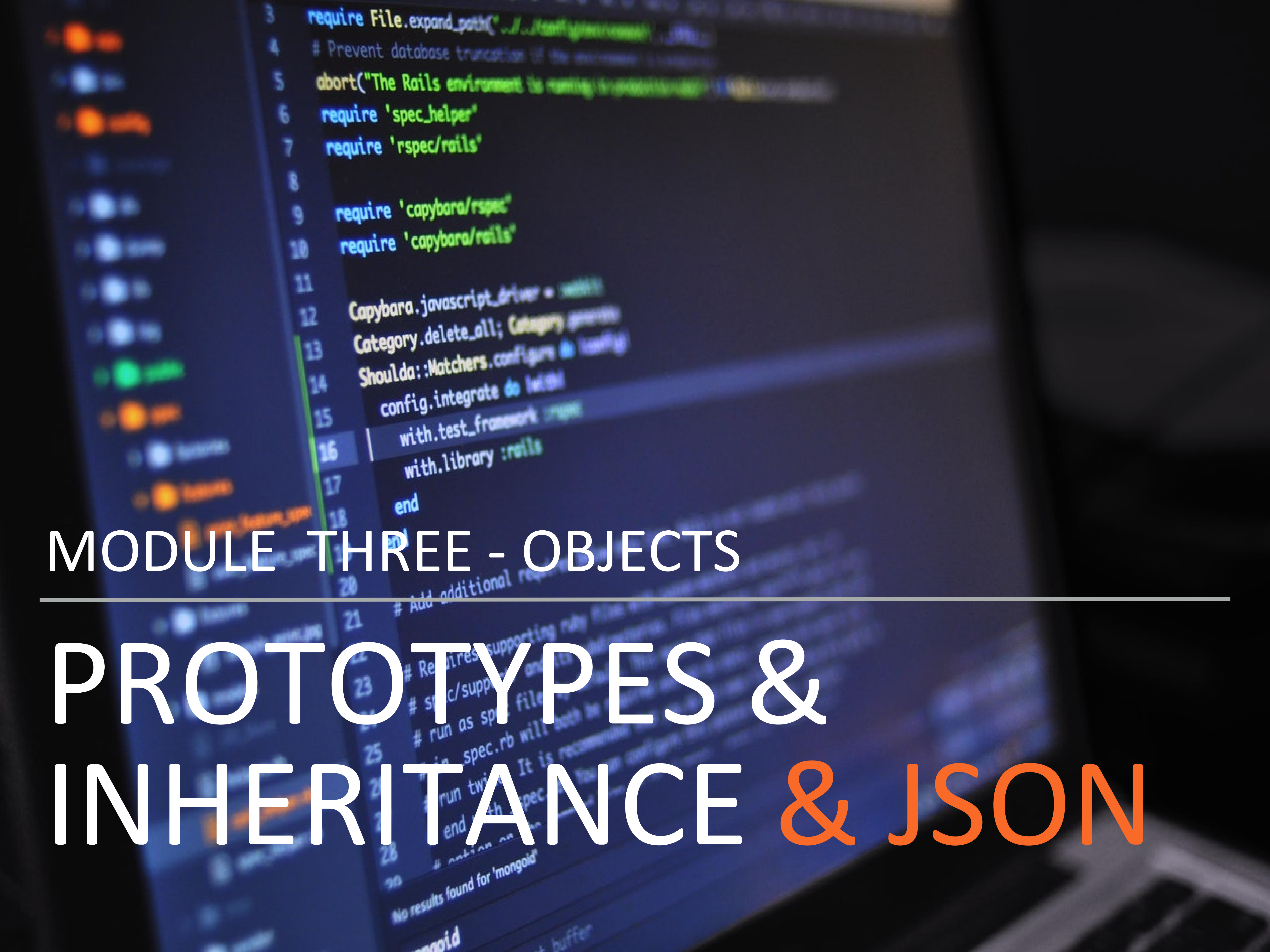


MODULE THREE - OBJECTS

PROTOTYPES & INHERITANCE & JSON

TODAY'S SCHEDULE

1. Cool Things
2. Review Constructor Functions
3. Prototypes & Inheritance
4. A Classier Way …
5. Do You Know JSON?
6. Recap, Weekly Tasks, Next Week

LEARNING OBJECTIVES

1. Explore and demonstrate understanding of object prototypes, the prototype chain, using the prototype property and inheritance

2. Explore and use ECMAScript 2015 class syntax

3. Employ the most common functions and methods used with current client-side JavaScript techniques;

4. Optimize code for increased functionality, performance, readability, and reusability

COOL THINGS!

```
require File.expand_path('../../config/environment', [illegible]
# Prevent database truncation [illegible]
abort("The Rails environment [illegible]
require 'spec_helper'
require 'rspec/rails'

require 'capybara/rspec'
require 'capybara/rails'

Capybara.javascript_driver = [illegible]
Category.delete_all; Category [illegible]
Shoulda::Matchers.configure do |config|
  config.integrate do |with|
    with.test_framework :rspec
    with.library :rails
  end
end

# Add additional requires below this line. [illegible]

# Requires supporting ruby files with [illegible]
# spec/support/ and its subdirectories. [illegible]
# run as spec files by default. This means [illegible]
# in _spec.rb will both be required and [illegible]
# run twice. It is recommended [illegible]
# end with _spec.rb. You can configure [illegible]
# option on the [illegible]
```

RESOURCES, LINKS TUTORIALS AND OTHER COOL THINGS…

- https://www.freecodecamp.org/news/is-vanilla-javascript-worth-learning-absolutely-c2c67140ac34/
- https://medium.com/better-programming/9-projects-to-inspire-front-end-developers-in-2020-a404545f6369
- https://learnvanillajs.com/
- https://dzone.com/articles/hiring-my-cat-as-a-software-developer

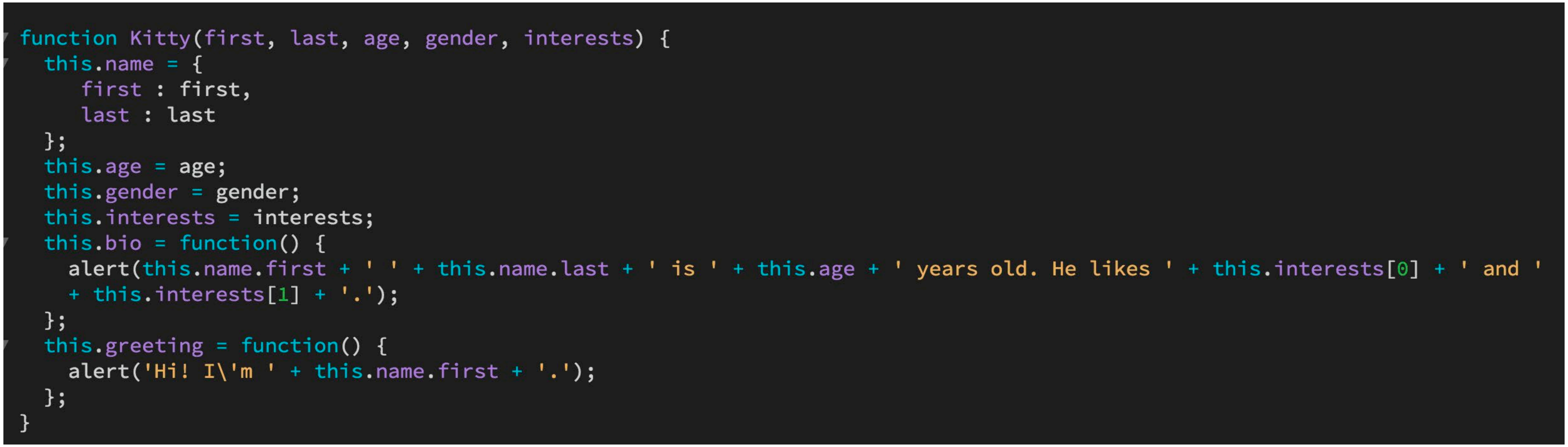
LET’S REVIEW OOJS

!IMPORTANT OOP CONCEPTS

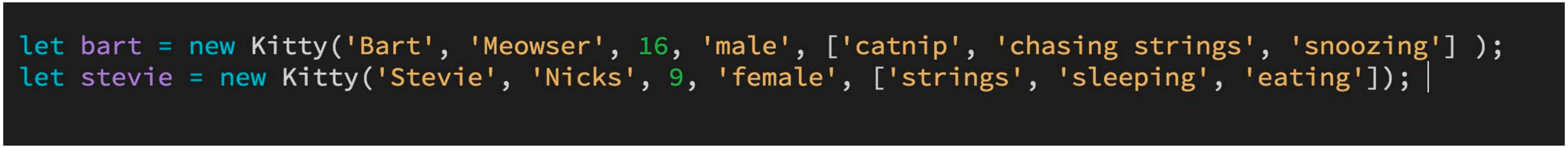
- Abstraction : creating a simple model of a more complex thing
- Encapsulation : our object becomes a container or capsule for properties and methods
- Instantiation: creating an object instance from a class
- Polymorphism: the ability for multiple objects to implement the same functionality

OOP + JS =

OOJS



```
function Kitty(first, last, age, gender, interests) {
  this.name = {
     first : first,
     last : last
  };
  this.age = age;
  this.gender = gender;
  this.interests = interests;
  this.bio = function() {
    alert(this.name.first + ' ' + this.name.last + ' is ' + this.age + ' years old. He likes ' + this.interests[0] + ' and '
    + this.interests[1] + '.');
  };
  this.greeting = function() {
    alert('Hi! I\'m ' + this.name.first + '.');
  };
}
```



```javascript
let bart = new Kitty('Bart', 'Meowser', 16, 'male', ['catnip', 'chasing strings', 'snoozing'] );
let stevie = new Kitty('Stevie', 'Nicks', 9, 'female', ['strings', 'sleeping', 'eating']);
```

WHAT'S GOING ON HERE?

- constructor functions help us to create object templates
- we can instantiate new object instances by calling the constructor function, using the new keyword & passing through the required arguments

CONSTRUCTOR FUNCTIONS

The constructor function allows us to create a class. It’s pretty much like a standard function, except we aren’t creating an empty object or returning it.

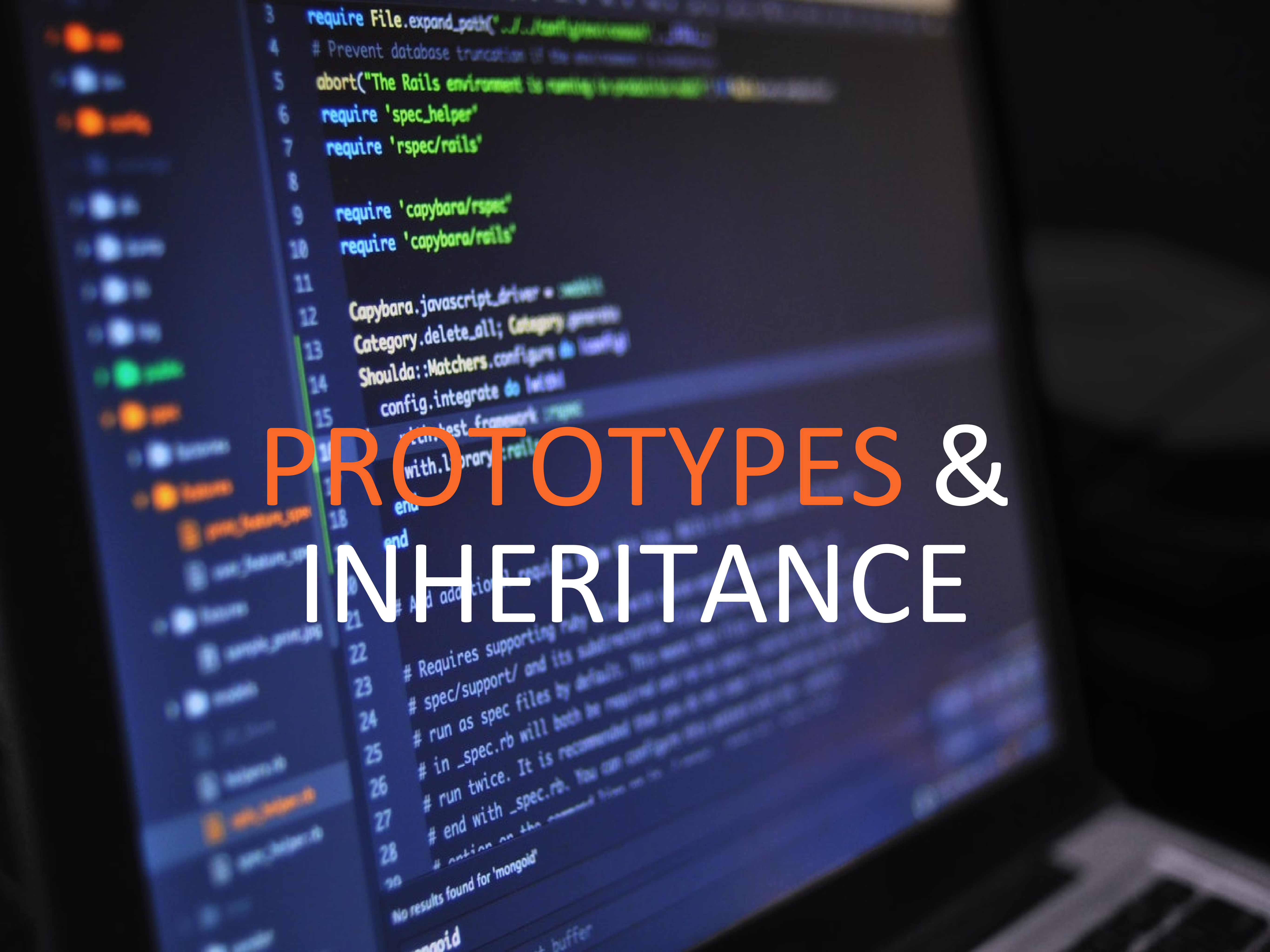
CONSTRUCTOR FUNCTIONS

Constructors help to give our code order. We can create constructors in one place, then create object instances when we need them.

COOL, BUT…

- one small drawback is that every time we call our constructor function, we are defining greeting( ) every time

- Don't worry, we'll improve on this when we talk about prototypes and inheritance.



```
require File.expand_path('../../config/environment', __FILE__)
# Prevent database truncation if the environment is production
abort("The Rails environment is running in production mode!") if Rails.env.production?
require 'spec_helper'
require 'rspec/rails'

require 'capybara/rspec'
require 'capybara/rails'

Capybara.javascript_driver = :webkit
Category.delete_all; Category.generate
Shoulda::Matchers.configure do |config|
  config.integrate do |with|
    with.test_framework :rspec
    with.library :rails
  end
end

# Add additional requires below this line. Rails is not loaded until this point!

# Requires supporting ruby files with custom matchers and macros, etc, in
# spec/support/ and its subdirectories. Files matching `spec/**/*_spec.rb` are
# run as spec files by default. This means that files in spec/support that end
# in _spec.rb will both be required and run as specs, causing the specs to be
# run twice. It is recommended that you do not name files matching this glob to
# end with _spec.rb. You can configure this pattern with the --pattern
# option on the command line or in ~/.rspec, .rspec or `.rspec-local`.
```

No results found for 'mongoid'

PROTOTYPES & INHERITANCE

A PROTOTYPE-BASED LANGUAGE

- JavaScript is often described as a prototype- based language
- prototypes are the mechanism by which objects inherit features

from one another in JS

HOW DOES THIS WORK?

- objects have a prototype object (acts a template object that it inherits methods and properties from)
- an object's prototype object may also have a prototype object, creating a prototype chain

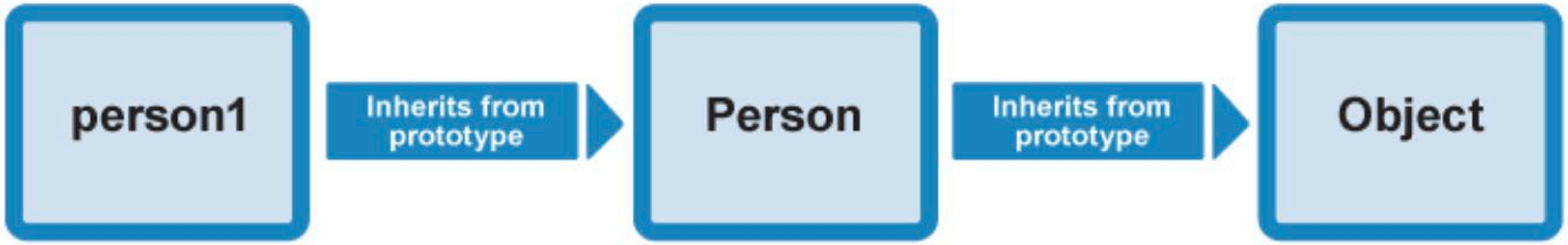
PROTOTYPAL INHERITANCE

- a link is made between the object instance and it's prototype, which is derived from

  the prototype property on the constructor

- the properties and methods are found by walking up the chain of prototypes

PROTOTYPAL INHERITANCE



SOURCE : https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object_prototypes

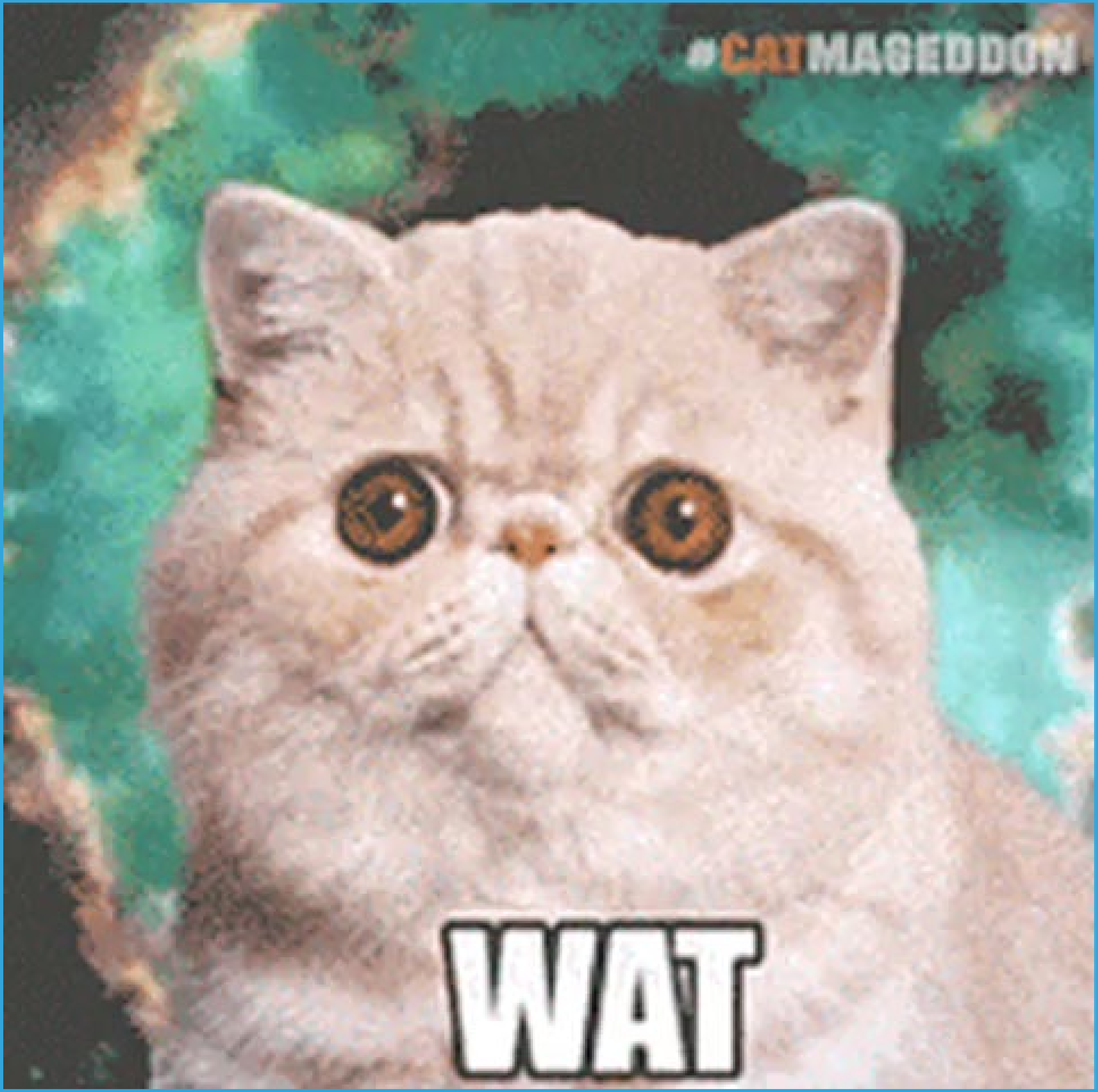
THE PROTOTYPE CHAIN

- When it comes to object prototypes, properties and methods are defined on the prototype property of the Object's constructor function, not the object instances themselves.
- Methods and properties are not copied from one object to another in the prototype chain.

MODIFYING PROTOTYPES

- We can modify the prototype property on our constructor function
- Good idea to add our methods to the prototype property object of our

  constructor function instead of inside the constructor function itself


#CATMAGEDDON
WAT

Weekly Learning > Week 9 > Code Examples >

Week 9 Start > Example One

A CLASSIER WAY

```
require File.expand_path(
# Prevent database truncation
abort("The Rails environment
require 'spec_helper'
require 'rspec/rails'

require 'capybara/rspec'
require 'capybara/rails'

Capybara.javascript_driver
Category.delete_all; Category
Shoulda::Matchers.configure do
  config.integrate do
    with.test_framework :rspec
    with.library :rails
  end
end

# Add additional requires below

# Requires supporting ruby
# spec/support/ and its
# run as spec files by default.
# in _spec.rb will both
# run twice. It is
# end with _spec.rb.
```

No results found for 'mongoid'

JAVASCRIPT GOT CLASSY!

- ECMAScript 2015 introduced classes to JavaScript
- cleaner and easier syntax, same old prototypal inheritance
- subclasses, getters and setters

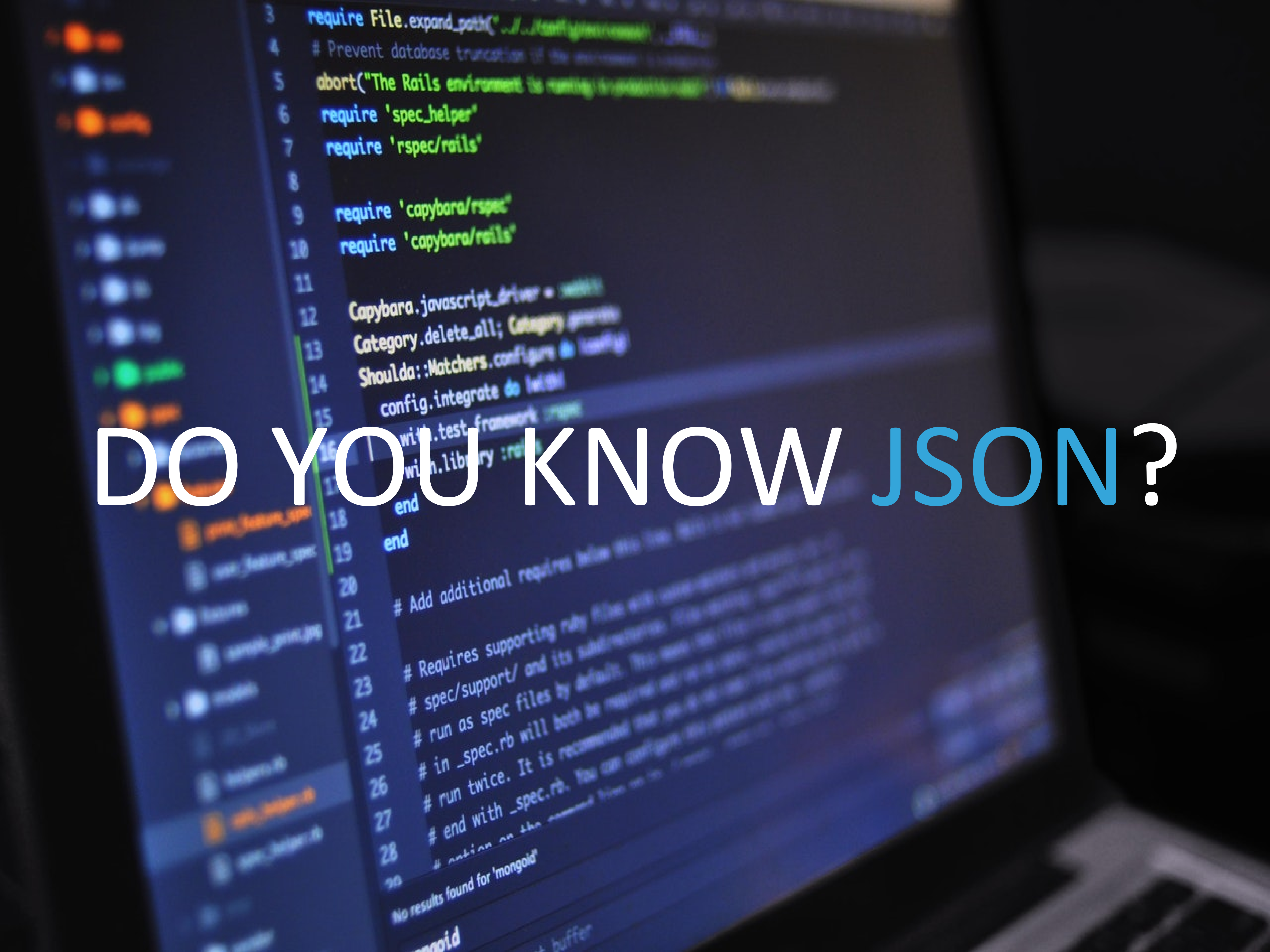
LETS THINK ABOUT THIS.

Watch a Video

Weekly Learning > Week 9 > Code Examples >

Week 9 Start > Example Two

DO YOU KNOW JSON?

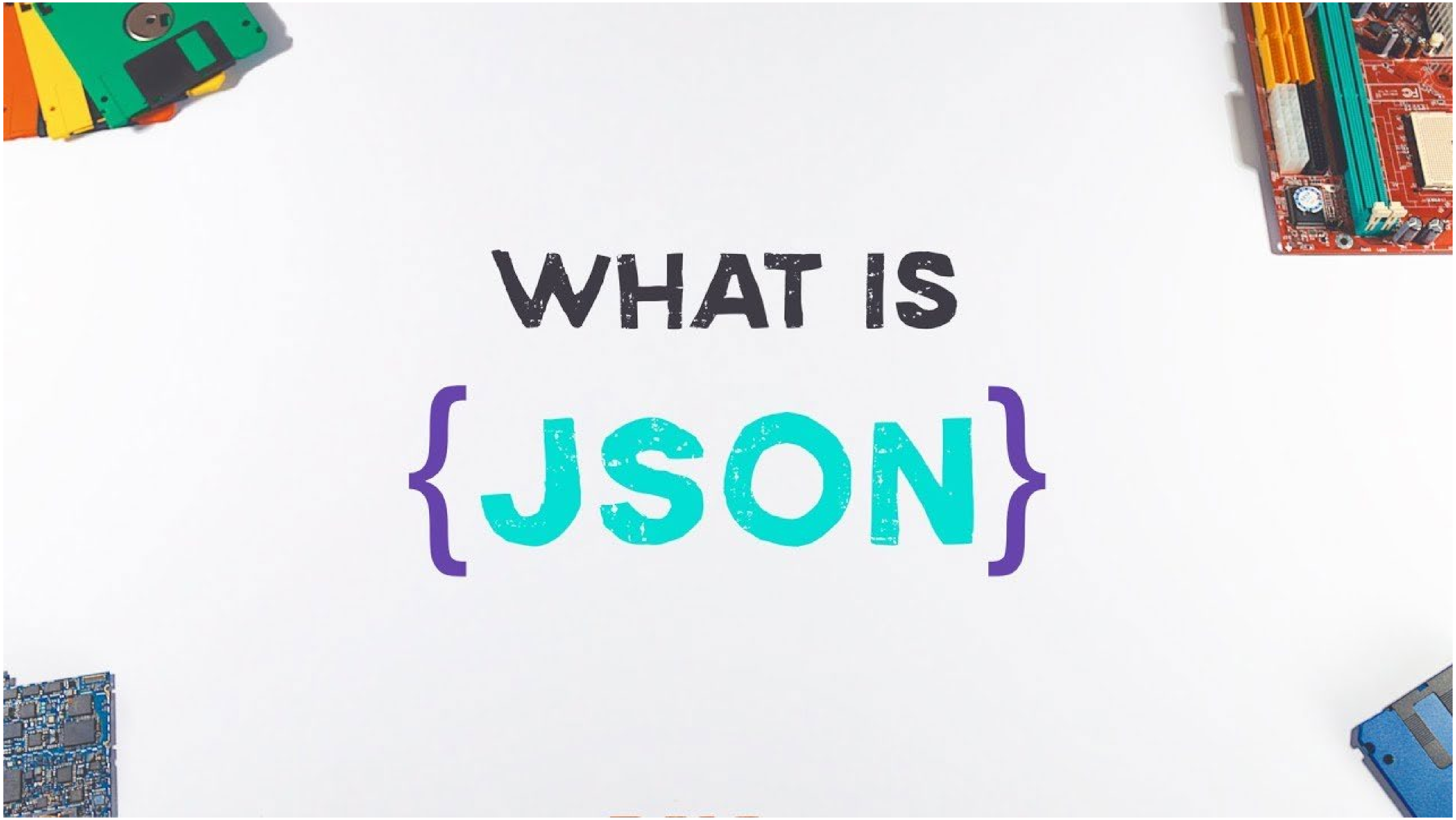


```
require File.expand_path([illegible]
# Prevent database truncation [illegible]
abort("The Rails environment [illegible]
require 'spec_helper'
require 'rspec/rails'

require 'capybara/rspec'
require 'capybara/rails'

Capybara.javascript_driver = [illegible]
Category.delete_all; [illegible]
Shoulda::Matchers.configure [illegible]
  config.integrate do [illegible]
    with.test_framework [illegible]
    with.library [illegible]
  end
end

# Add additional requires below [illegible]

# Requires supporting ruby files [illegible]
# spec/support/ and its [illegible]
# run as spec files by default. [illegible]
# in _spec.rb will both be [illegible]
# run twice. It is [illegible]
# end with _spec.rb. You can [illegible]
```

No results found for 'mongoid'

JSON

- stands for JavaScript Object Notation
- separate from JavaScript
- text-based data format following JavaScript object syntax
- used for transmitting data in web applications


WHAT IS
{JSON}

Weekly Learning > Week 9 > Code Examples >

Week 9 Start > Example Three

RECAP, THIS WEEK, NEXT WEEK

OOP

- we use objects to model real world things in our programs/ provide easy access to functionality
- objects contain related data and code which represent info or functionality

OOJS

- We can create define object templates using a constructor function (like a class in JS) or we can use class syntax
- Either way inheritance is prototype-based

PROTOTYPES & INHERITANCE

- prototypes and inheritance are complex, but they provide a lot of power and flexibility
- likely only use for larger projects
- don’t have too many levels of inheritance and keep track of where you define your methods and properties

JSON

- stands for JavaScript Object Notation
- separate from JavaScript
- text-based data format following JavaScript object syntax
- used for transmitting data in web applications

NEXT WEEK - APIs!!!!!!

THANKS TO (SOURCES):

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object_prototypes

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Inheritance

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON#:~:text=JavaScript%20Object%20Notation%20(JSON)%20is,page%2C%20or%20vice%20versa).