Week 6
Modules: 6

SaaS Client:
Mobile and Desktop SaaS clients, JavaScript

Topics:

- JavaScript Syntax
- Document Object Model
- jQuery, Callbacks, Event Loops, AJAX
- JavaScript in Rails + Bonus Exam Question

Background

- Tim Berners-Lee (1990) - Implemented first HTTP client and server.
- HTTP - Hypertext Transfer Protocol; Hypertext is a system that makes it easy to access related info and graphics via clickable links. Foundation of the web.
- Mosaic and Netscape Navigator - first few web browsers.
- Early web pages were pretty basic - mainly static, non-interactive text and graphics.

Static vs Dynamic Websites

- ***Static websites:*** Display the same content for every user. Site is mainly authored in HTML. eg. A personal portfolio website.
- ***Dynamic website:*** Different content displayed for each user. Many technologies used, but the browser still renders HTML. Eg. Twitter.

Enter JavaScript

JS Intro

- JavaScript is to Java as car is to carpet.
- Initially called LiveScript.
- JavaScript name intended to capitalize on Java's popularity.
- Initially developed by Brendan Eich (while at Netscape)

JS Intro (cont.)

- Created initially for Netscape Navigator to make web pages more interactive.
- Dynamically typed just like Ruby. First class functions like Scheme.
- JavaScript today is used beyond the browser.

Big Picture

Four Major Approaches to using JavaScript (least to most JS intensive):

1. Add JavaScript to HTML/CSS pages for enhanced user experiences
2. Single Page Applications: Each “page” is managed by JS, content updated in-place.
3. Write full client side applications (i.e. Google Docs), could also work offline
4. Server side apps, like Rails, but with JS frameworks (i.e. Node, Express)

Other ways to use JS

- Build a cross-platform desktop application like Slack using JavaScript, HTML and CSS. Slack essentially packages a custom version of Chrome (“Electron”) that manages interactions between the app and your OS.
    - This codebase is the same as the Slack browser client.
- Build cross-platform mobile clients using JavaScript, HTML & CSS instead of native OS technologies like Swift on iOS.
    - frameworks like Ionic (basically a purpose-built browser) and React Native (compile JS to native Swift / Java)

Unobtrusive JS

When enhancing apps w/ JS, abide by **unobtrusive JavaScript**.

Unobtrusive JavaScript means:

1. Site should remain accessible and browser compatible despite enhancements. Accessibility ("a11y") and compatibility could largely be addressed by using libraries like jQuery
   a. This is a component of *progressive enhancement.*
2. Separate HTML markup (content) from JavaScript (behavior). Instead of mixing JS and HTML in same file, use separate files for each.

JS vs Ruby

	Ruby	JavaScript
Runs primarily on	Server	Browser
Framework (for CS169)	Rails	jQuery (more of a library than framework)
Role in Client-Server architecture	Controller receives request, interacts with DB via model and renders the page (view).	Browser fetches JS file from server. JS code executed on specific events eg clicks. JS fetches data from Rails app and updates the browser.

JS vs Ruby (cont.)

	Ruby	JavaScript
Debugging	Byebug; Rails console;	Browser Console (“Web Inspector”)
Testing	rspec (via rspec-rails)	jasmine (Unit Testing for JS)
Linter (check for code smells & code quality)	Rubocop	JSLint or ESLint
Specify list of third-party dependencies	Gemfile	package.json

JS vs Ruby (cont.)

	Ruby	JavaScript
Interpreters	MRI ("Matz's Ruby Interpreter") aka CRuby — the "official" Ruby release.	NodeJS used to run JS programs outside the browser. V8 in Chrome, JavaScriptCore (Safari, iOS, macOS)
Program used to install third-party dependencies	`bundle` (bundler)	2 main options: `npm` - Default for NodeJS `yarn` - Default for Rails apps
Pinning third party dependencies	Gemfile.lock	npm - package-lock.json yarn - yarn.lock

ECMAScript

- ECMA = European Computer Manufacturers' Association, a standards body.
- ECMAScript (ES): The standard that defines the JavaScript (JS) language.
  - Oracle owns the JavaScript name, but there is no standard implementation.
- ECMAScript versions: e.g. ES5 released in 2009 vs newer ES6/ES2015 (2015), today: ES2021
  - Every browser implements a different subset of features

JS Syntax

JS Types

- Two main categories of types in JS
- Primitive types and Object types
- Primitives:
    a. String
    b. Number - Represents Integers and 64-bit doubles
    c. undefined - uninitialized variables & non-existent fields in objects.
    d. null - different from undefined. Can be assigned to variable. Similar to Ruby nil.
    e. BigInt - arbitrary magnitude integers
    f. Boolean
    g. Symbol (similar to Ruby symbols, but used for hiding information)

JS Objects

- Most compound types in JavaScript is the Object type
- Object used to represent classes with fields as well as dictionaries
- Object in JavaScript is similar to Ruby’s hash.
- Example Object:

```
movie = { title: ‘Avatar’, releaseYear: 2009, rating: ‘PG-13’ }
```

- `movie.title` prints “Avatar”
- `movie[“title”]` also prints “Avatar”
- Syntax used to create JS Objects is called JSON - JavaScript Object Notation
- JSON is a common wire format for the web. Alternatives: XML & protobuf.

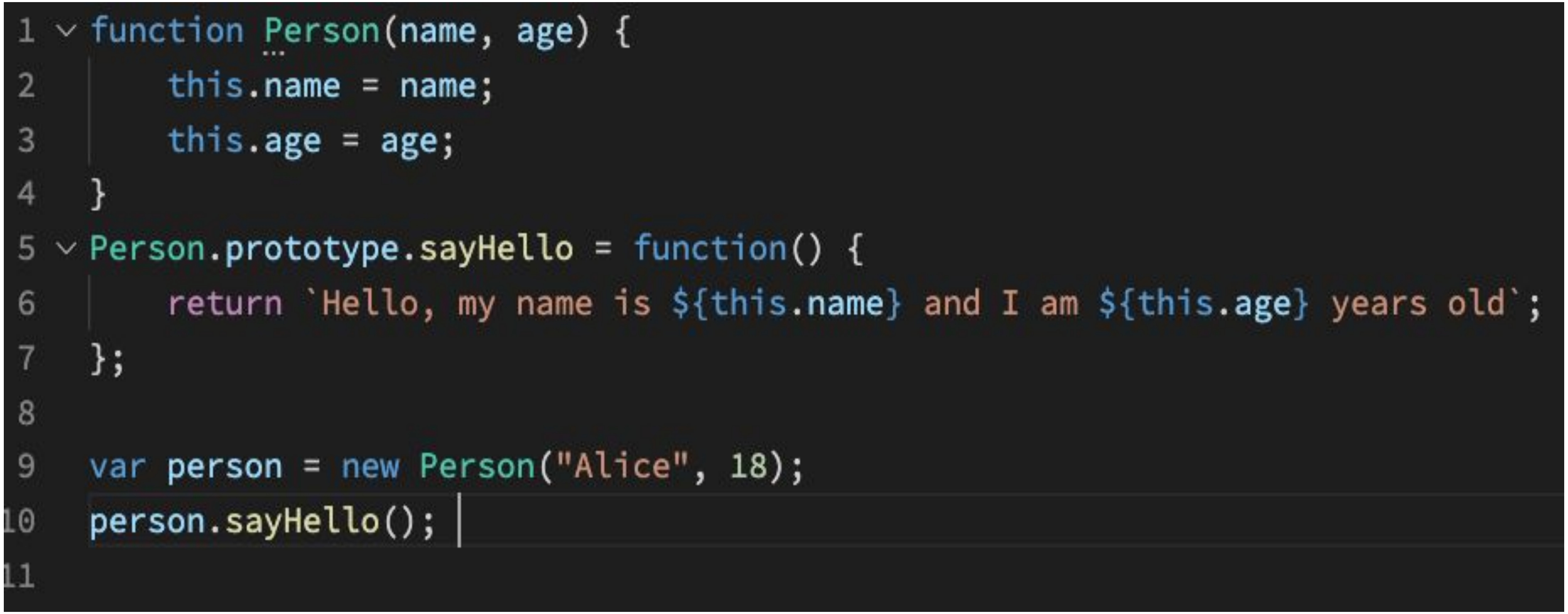
JS Control Flow

- == is the weak equality operator. Attempts to type-cast.
- ‘1’ == 1 returns true
- === is the strict equality operator. Operands only equal if their types match.
- ‘1’ === 1 returns false but 1 === 1 returns true.
- Common Flow control operators:
  - while (condition) { }
  - for(;;) { }
  - if ... else if ... else
  - [condition] ? [true case] : [false case]
  - switch/case
  - try/catch/throw
  - return
  - break

JS Control Flow (cont.)

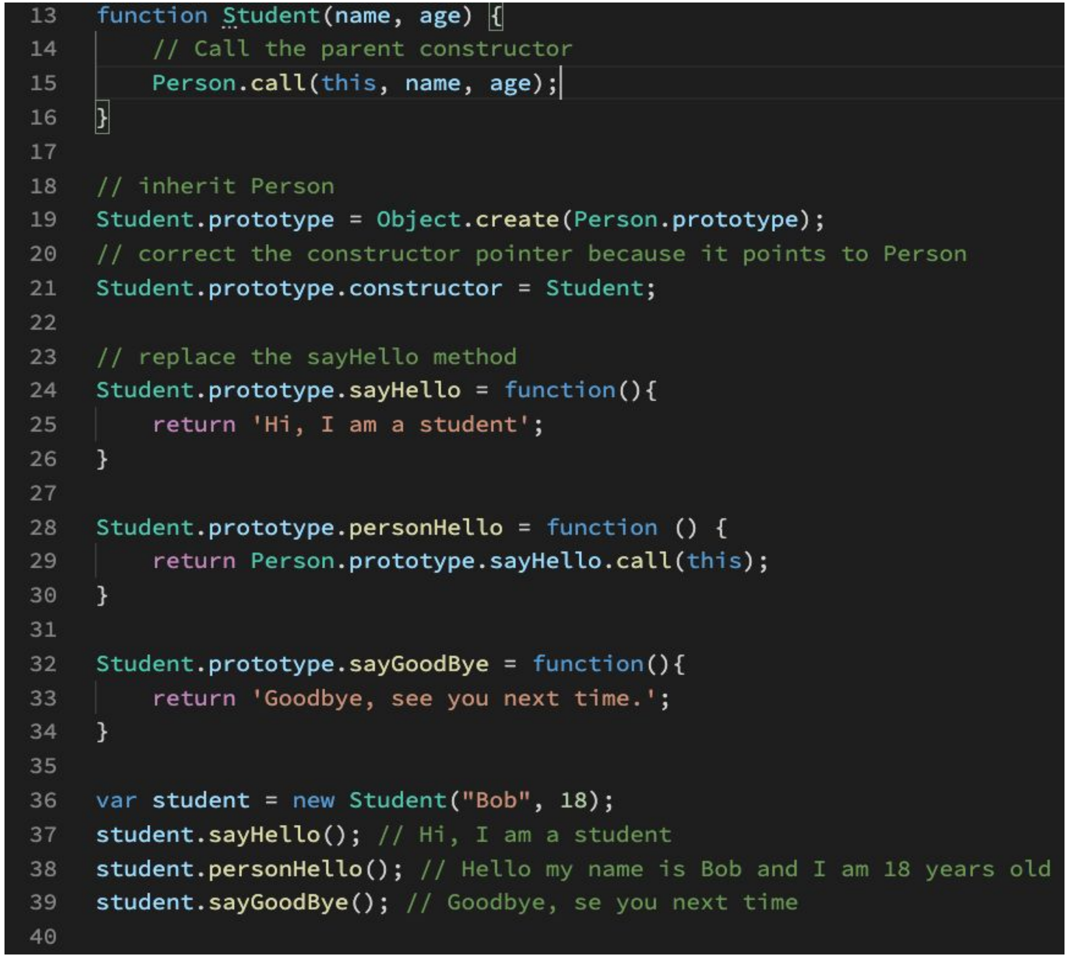
- JS supports Object Oriented Programming
- In older es5 syntax, OOP was done using prototypes.
- Prototype: Mechanism through which objects inherit features from each other.
- Classes are declared as functions
- Such functions start with upper-case letter by convention and act as constructor
- Methods are added to the .prototype field
- New keyword used to initialize instances of class.
- Each class in inheritance hierarchy has its own .__proto__ field.

Prototypes



```javascript
function Person(name, age) {
    this.name = name;
    this.age = age;
}
Person.prototype.sayHello = function() {
    return `Hello, my name is ${this.name} and I am ${this.age} years old`;
};

var person = new Person("Alice", 18);
person.sayHello();
```

Prototypes (cont.)

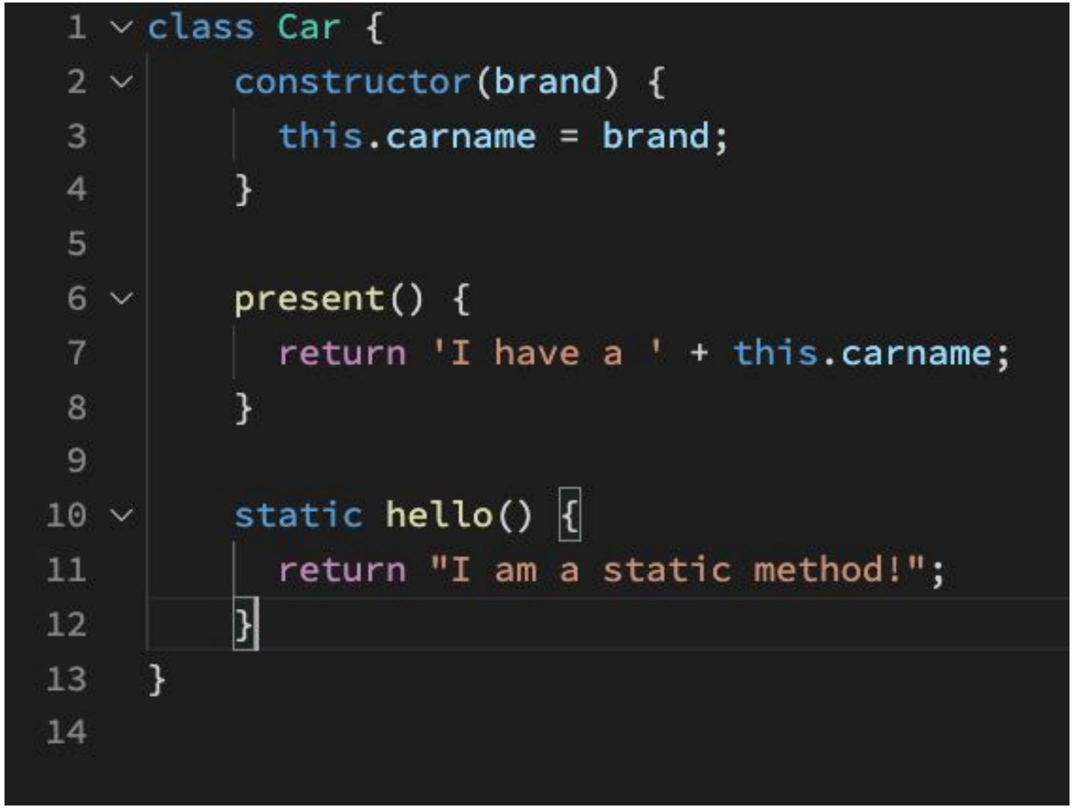


```
function Student(name, age) {
    // Call the parent constructor
    Person.call(this, name, age);
}

// inherit Person
Student.prototype = Object.create(Person.prototype);
// correct the constructor pointer because it points to Person
Student.prototype.constructor = Student;

// replace the sayHello method
Student.prototype.sayHello = function(){
    return 'Hi, I am a student';
}

Student.prototype.personHello = function () {
    return Person.prototype.sayHello.call(this);
}

Student.prototype.sayGoodBye = function(){
    return 'Goodbye, see you next time.';
}

var student = new Student("Bob", 18);
student.sayHello(); // Hi, I am a student
student.personHello(); // Hello my name is Bob and I am 18 years old
student.sayGoodBye(); // Goodbye, se you next time

```

ES6

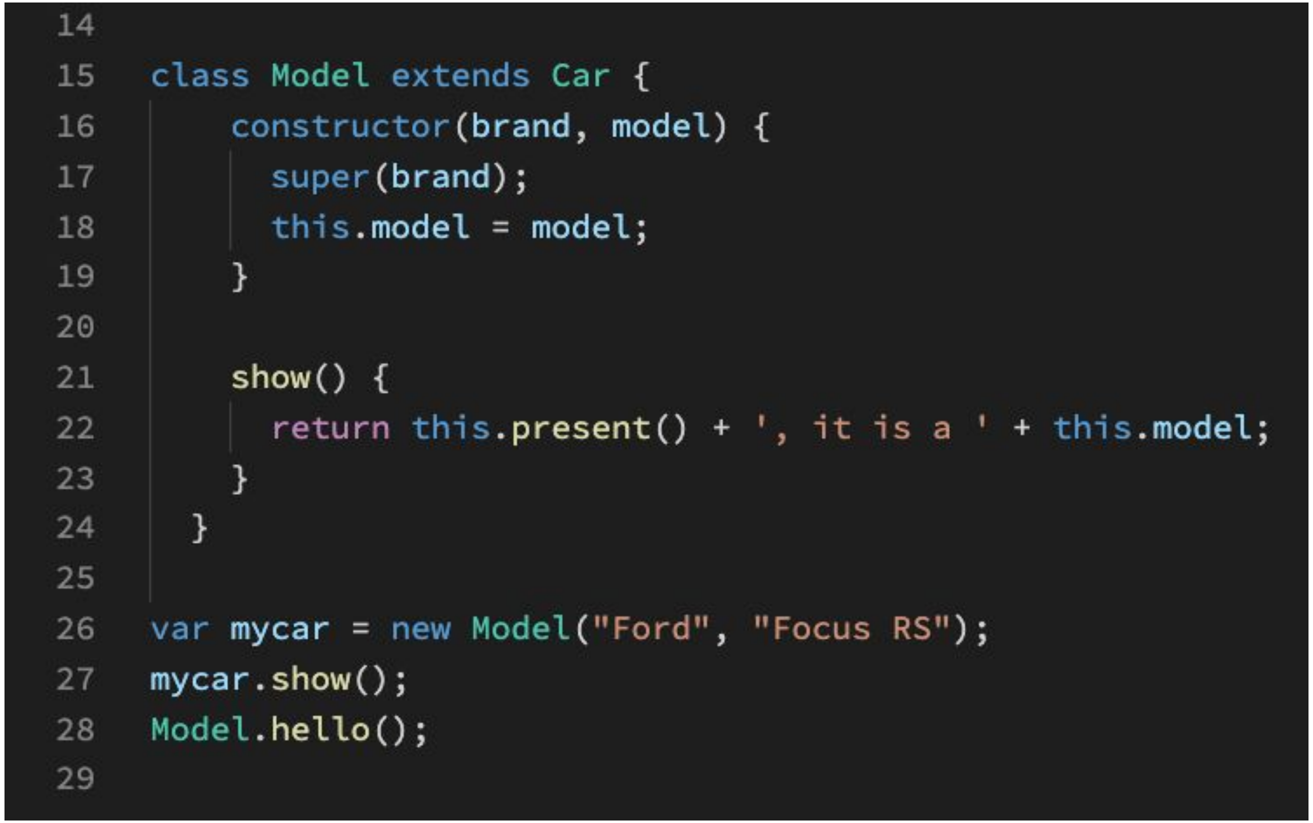
- ES6 introduced the class keyword
- Class is just syntactic sugar for prototype syntax
- Notice that prototype inheritance syntax is a bit contrived
- ES6 made it easier to declare classes
- To ease transition between ES5 to ES6 as browsers added support, we use "JavaScript compilers" like BabelJS

ES6 Classes



```
class Car {
    constructor(brand) {
      this.carname = brand;
    }

    present() {
      return 'I have a ' + this.carname;
    }

    static hello() {
      return "I am a static method!";
    }
}
```

ES6 Classes (cont.)



```
class Model extends Car {
    constructor(brand, model) {
      super(brand);
      this.model = model;
    }

    show() {
      return this.present() + ', it is a ' + this.model;
    }
  }

var mycar = new Model("Ford", "Focus RS");
mycar.show();
Model.hello();

```

Yet another JS vs Ruby : syntax comparison edition

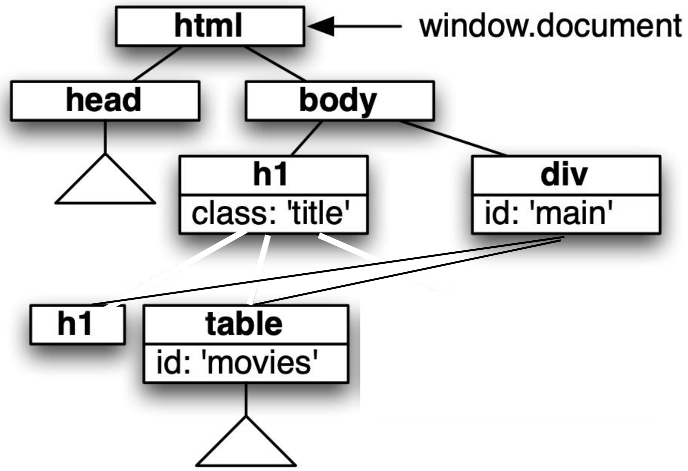
	Ruby	JavaScript
Declare functions	def hello	function hello()
Truthy/falsey values	nil, false are falsey. Zero, empty string, empty array, empty hash are truthy	0, empty strings considered as falsey.
Access other methods	self	this

DOM

(Document Object Model)

The DOM

A language agnostic + hierarchical representation of displayed content that allows programmatic access and update of a page.



```
<html>
  <head>
    <title>...</title>
    <script>...</script>
  </head>
  <body>
    <h1 class='title'>Rotten Potatoes!</h1>
    <div id='main'>
       <h1>All Movies</h1>
       <table id='movies'>
          ...
       </table>
    </div>
  </body>
</html>
```

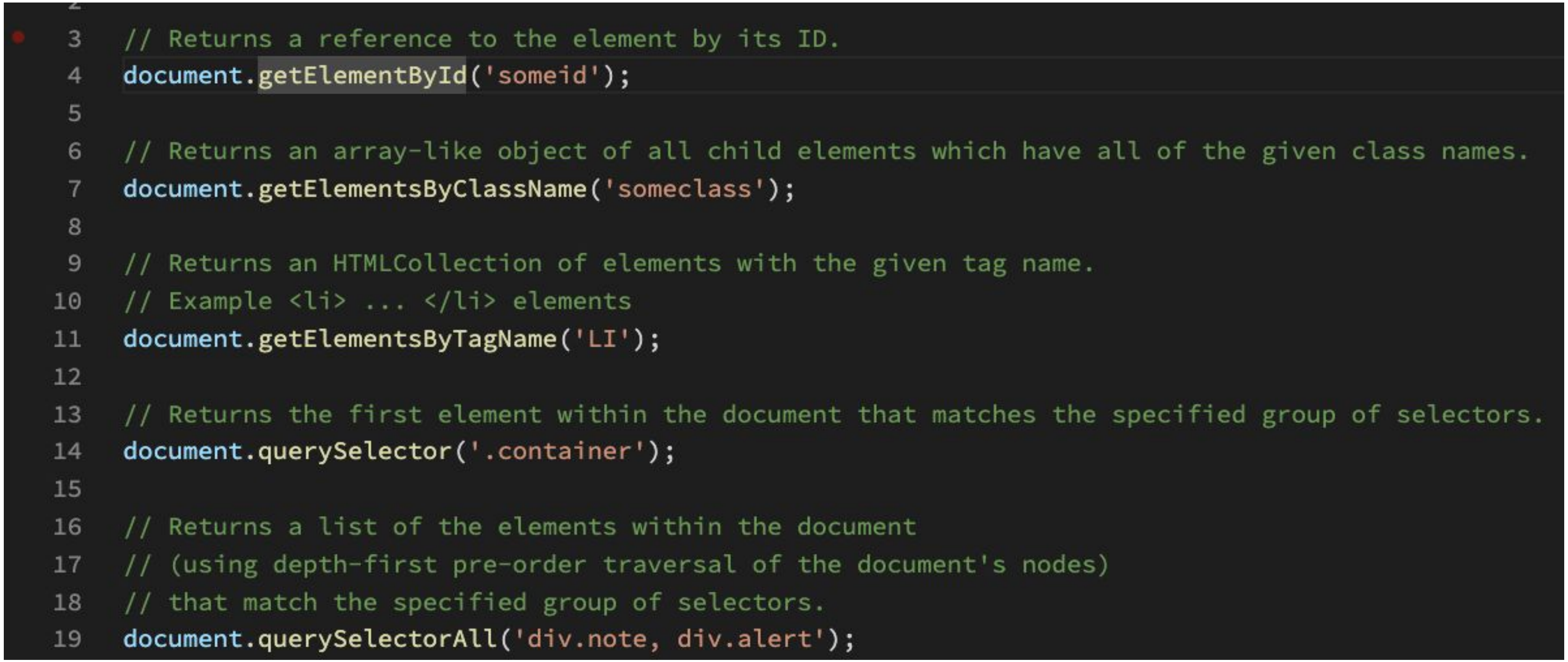
The DOM (cont.)

- Browsers provide an API to access the DOM
- Example query:

```
let el = document.getElementById("main-section"); //
```

- Above query will find the an element with attribute id="main-section" in the tree of elements displayed on the page.
- Note: no two elements should have the same id on the same page.
- jQuery was introduced to provide an easy to use DOM API that hides differences in between browsers.

DOM Query API

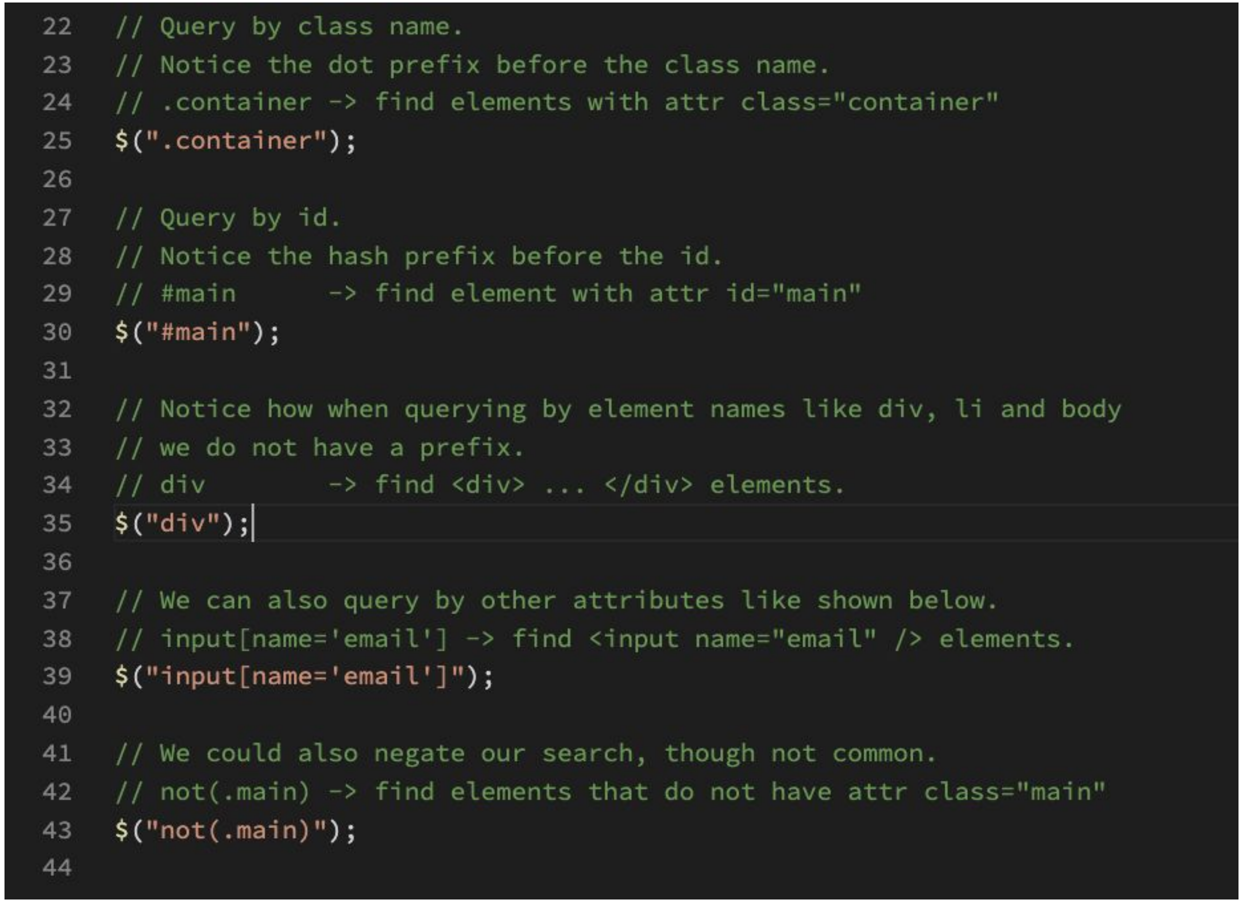


```javascript
// Returns a reference to the element by its ID.
document.getElementById('someid');

// Returns an array-like object of all child elements which have all of the given class names.
document.getElementsByClassName('someclass');

// Returns an HTMLCollection of elements with the given tag name.
// Example <li> ... </li> elements
document.getElementsByTagName('LI');

// Returns the first element within the document that matches the specified group of selectors.
document.querySelector('.container');

// Returns a list of the elements within the document
// (using depth-first pre-order traversal of the document's nodes)
// that match the specified group of selectors.
document.querySelectorAll('div.note, div.alert');
```

jQuery

- We could use jQuery to query objects on the page.
- Typically jQuery library is loaded and available via the $ variable.
- Example of querying element by id using jQuery:
- $(‘#main-section);
- Finds element with id=”main-section” on the page.

jQuery (cont.)



```
// Query by class name.
// Notice the dot prefix before the class name.
// .container -> find elements with attr class="container"
$(".container");

// Query by id.
// Notice the hash prefix before the id.
// #main      -> find element with attr id="main"
$("#main");

// Notice how when querying by element names like div, li and body
// we do not have a prefix.
// div        -> find <div> ... </div> elements.
$("div");

// We can also query by other attributes like shown below.
// input[name='email'] -> find <input name="email" /> elements.
$("input[name='email']");

// We could also negate our search, though not common.
// not(.main) -> find elements that do not have attr class="main"
$("not(.main)");
```

Callbacks and Event Loop

- Motivation: when user interacts with elements on the page, we would like to update the page. Eg. When you select a state say California in a dropdown we want to update the successive dropdown to only show the list of counties in California.
- Events refer to changes in the DOM such as when user mouse moves to a new element or user clicks a button or checkbox.
- Callback are functions that are setup to be run when certain event

Callbacks and Event Loop (cont.)

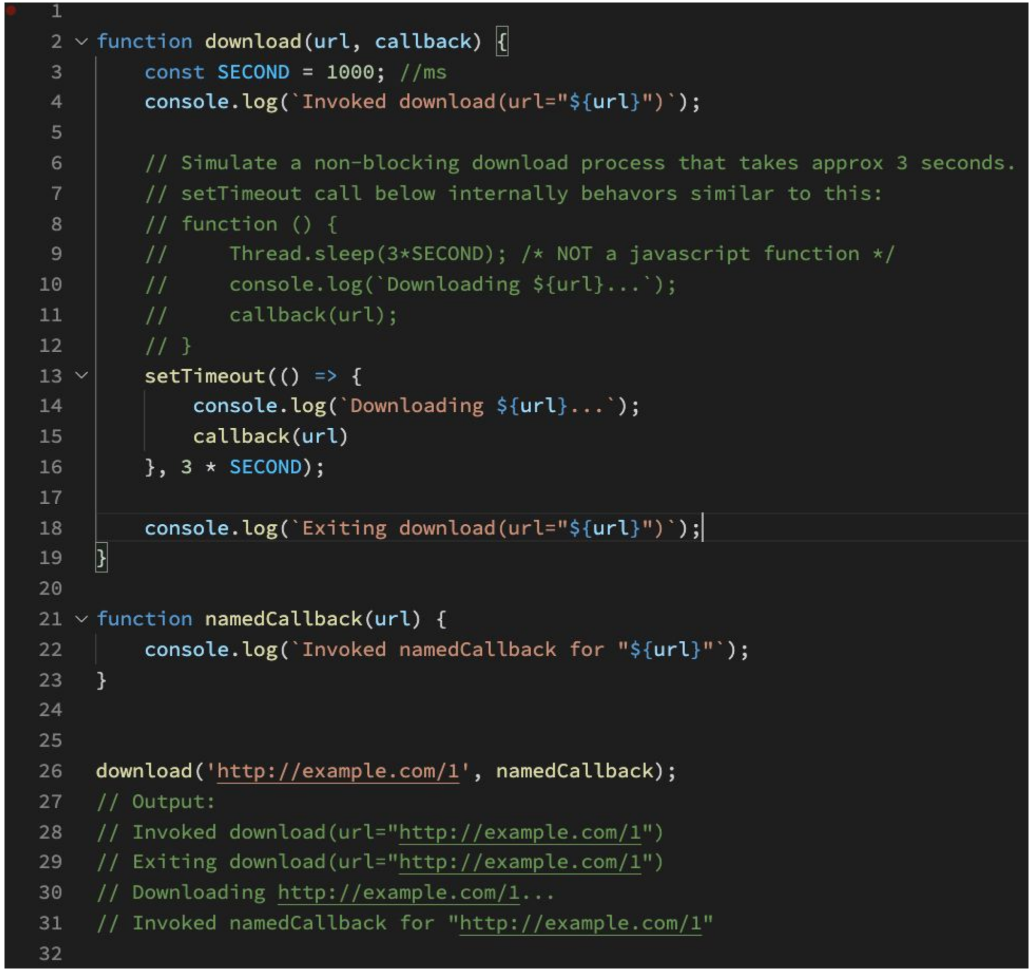
- Event loop = a paradigm in which programs wait for events to dispatch functions.
- The browser waits for user initiated events to invoke functions for different events set up using javascript.
- Example of setting up event callback using jQuery

```
element.on(‘eventname’, function() {
    // code to execute
}
```

Callbacks and Event Loop (cont.)

- Callbacks are thus functions that are passed into other functions as arguments with the intention to execute them later.
- In the example code to follow we will set up an example event loop.
- We use the setTimeout built in JavaScript function
- setTimeout(callback, waitInMs)
- It waits the given number of milliseconds before running the specified function
- This simulates a non-blocking callback since the code immediately after the call to setTimeout is executed without waiting for setTimeout() call to finish

Callbacks and Event loop (cont.)

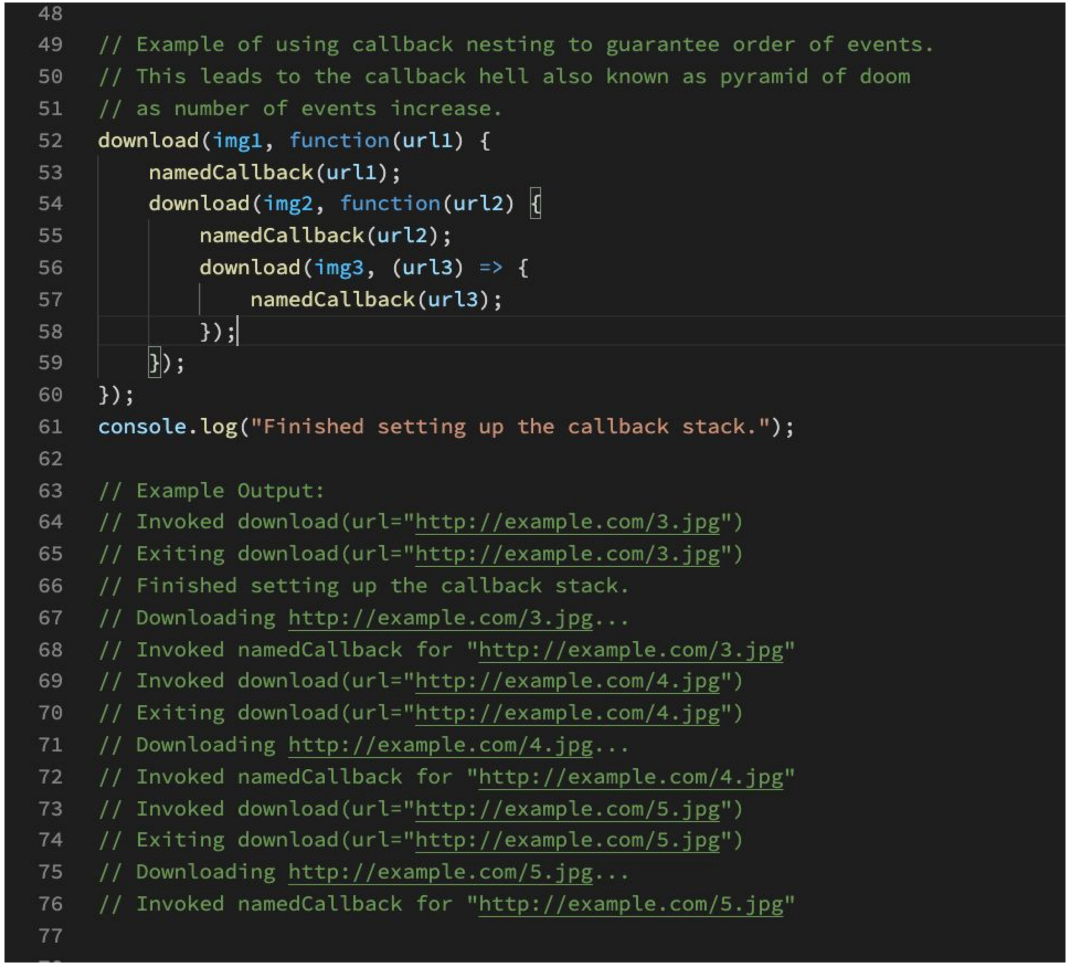


```

function download(url, callback) {
    const SECOND = 1000; //ms
    console.log(`Invoked download(url="${url}")`);

    // Simulate a non-blocking download process that takes approx 3 seconds.
    // setTimeout call below internally behavors similar to this:
    // function () {
    //     Thread.sleep(3*SECOND); /* NOT a javascript function */
    //     console.log(`Downloading ${url}...`);
    //     callback(url);
    // }
    setTimeout(() => {
        console.log(`Downloading ${url}...`);
        callback(url)
    }, 3 * SECOND);

    console.log(`Exiting download(url="${url}")`);
}

function namedCallback(url) {
    console.log(`Invoked namedCallback for "${url}"`);
}


download('http://example.com/1', namedCallback);
// Output:
// Invoked download(url="http://example.com/1")
// Exiting download(url="http://example.com/1")
// Downloading http://example.com/1...
// Invoked namedCallback for "http://example.com/1"

```

Callback Hell

- If we want to have a series of events occur in a specified order in the event loop, we may have to set up nested callbacks.
- For example:
- We want to download images in a specific order
- When the number of sequential events grows, we end up with a structure commonly referred to as the pyramid of doom/callback hell
- Promises and observables help alleviate this.

Callback Hell (cont.)

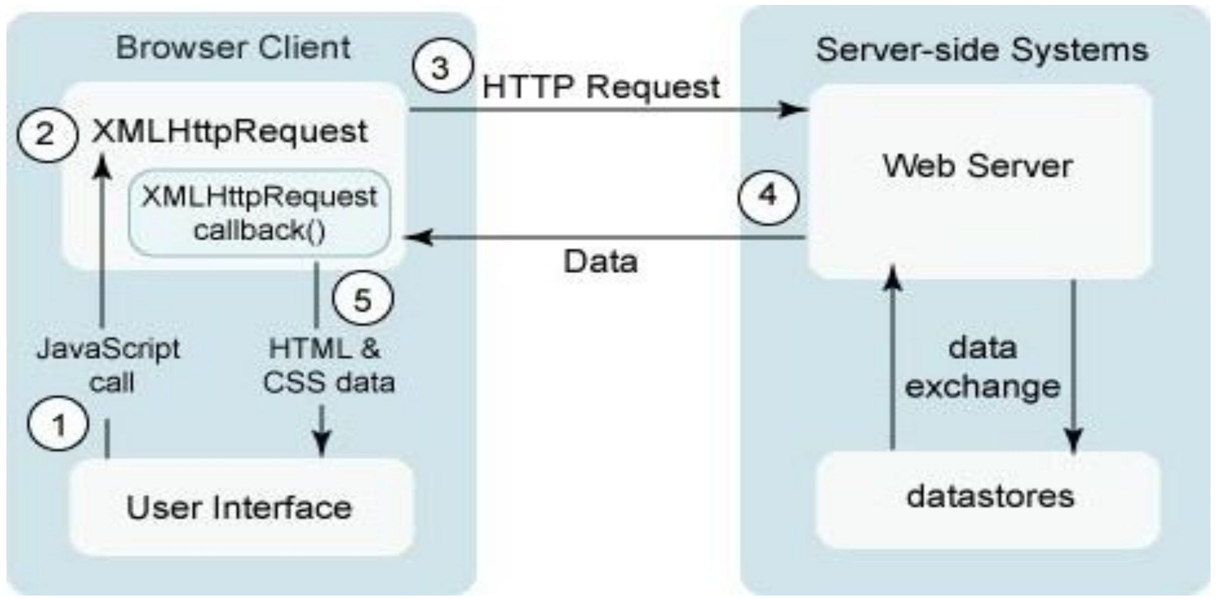


```
// Example of using callback nesting to guarantee order of events.
// This leads to the callback hell also known as pyramid of doom
// as number of events increase.
download(img1, function(url1) {
    namedCallback(url1);
    download(img2, function(url2) {
        namedCallback(url2);
        download(img3, (url3) => {
            namedCallback(url3);
        });
    });
});
console.log("Finished setting up the callback stack.");

// Example Output:
// Invoked download(url="http://example.com/3.jpg")
// Exiting download(url="http://example.com/3.jpg")
// Finished setting up the callback stack.
// Downloading http://example.com/3.jpg...
// Invoked namedCallback for "http://example.com/3.jpg"
// Invoked download(url="http://example.com/4.jpg")
// Exiting download(url="http://example.com/4.jpg")
// Downloading http://example.com/4.jpg...
// Invoked namedCallback for "http://example.com/4.jpg"
// Invoked download(url="http://example.com/5.jpg")
// Exiting download(url="http://example.com/5.jpg")
// Downloading http://example.com/5.jpg...
// Invoked namedCallback for "http://example.com/5.jpg"
```

AJAX

- **Motivation**: Instead of making the user wait for a request to be processed (which leads to “freezing” the UI), the request is processed *asynchronously* (hence, AJAX!)
- The webpage and global javascript environment are run on a single thread
- The programmer cannot spawn a new thread to move tasks away from the main “UI thread” since browser javascript does not expose any threading utilities.
- Hence javascript referred to as “single threaded” from the dev perspective
- AJAX provides a mechanism to move network calls away from UI thread to avoid freezing the page using the XmlHttpRequest class which deals with all async server calls

AJAX Requests



Testing JavaScript & AJAX

To test AJAX effectively, we use **Jasmine**, a TDD framework similar to RSpec.

AJAX Testing Core Problems

1. "Stub out the Internet" by intercepting AJAX calls. Instead, we test JS isolated from the server by returning predefined responses.
2. Use fixtures to test JS code relying on presence of certain DOM elements on a page.

See 6.8 for extensive documentation + examples of Jasmine in action.

```
...
describe('when successful server call', function() {
    beforeEach(function() {
        let htmlResponse =
            readFixtures('movie_info.html');
        spyOn($, 'ajax').and.callFake(
            function(ajaxArgs) {
                ajaxArgs.success(htmlResponse , '200');
            }
        );
        $('#movies a').trigger('click');
    });
...
```

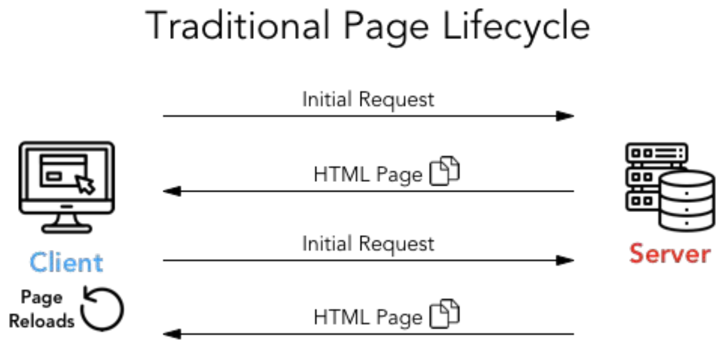
JS Apps

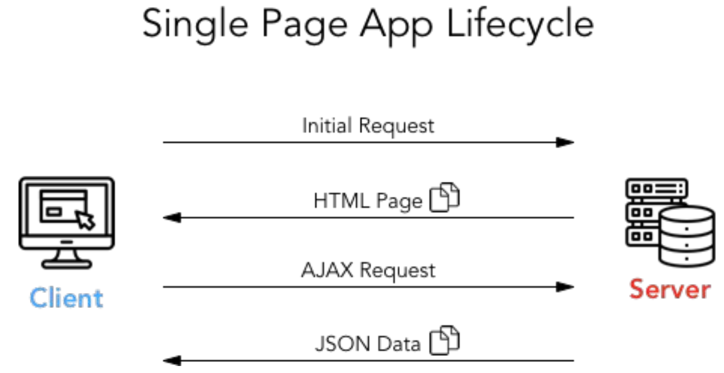
Single Page Apps

- After initial page load, all interactions occur without any page reloads
- Client side code requests "raw" data (i.e. JSON, text) instead of partial HTML page

To use JSON in client side code...

- Server should generate JSON responses
- Client must specify it expects JSON response in headers
- Client can use response to modify DOM
- For testing AJAX requests, use Jasmine to stub out the server





JavaScript in Rails

Using JS in Rails

Two ways to integrate JS with Rails:

1. Webpacker
2. Sprockets

As of Rails > 6.0.0, Webpacker replaced Sprockets as the new standard for writing JS.

Using JS in Rails (cont.)

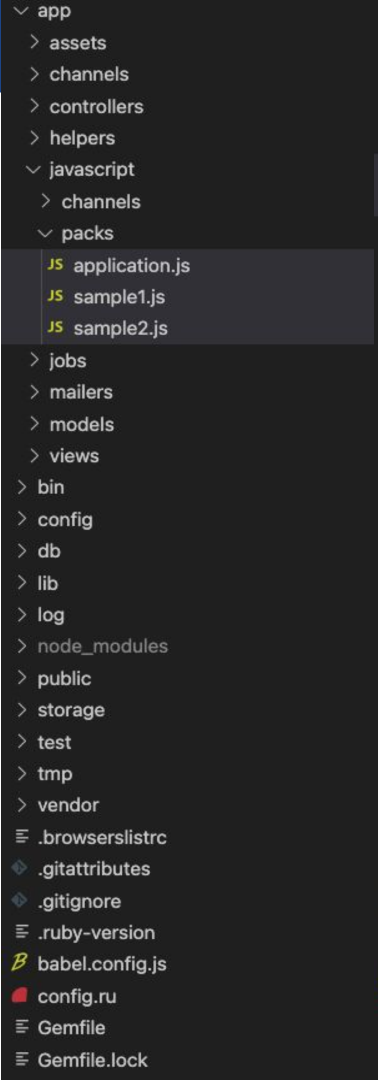
Where does the code go?

With Webpacker all the JS code lived under /app/javascript/packs, which is automatically created with Rails 6.

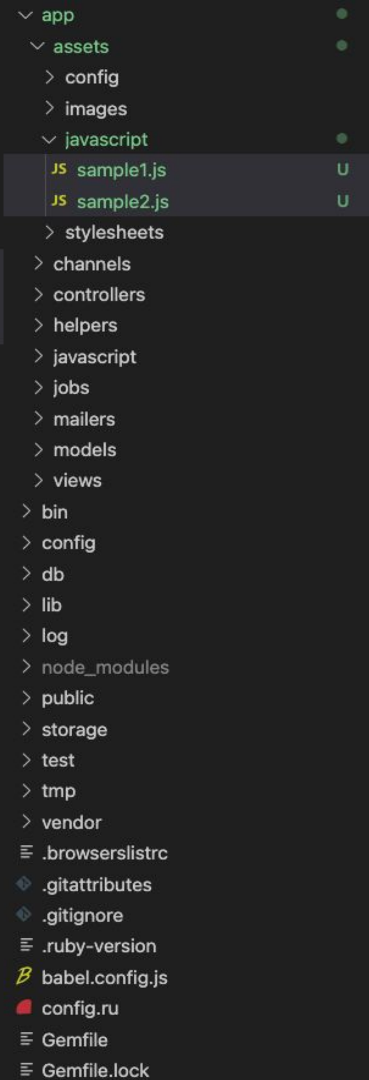
Use: `<%= javascript_pack_tag ‘hello_js’ %>`

With Sprockets, JS code lives under /app/assets/javascript which is not initialized be default in Rails 6.

Use: `<%= javascript_include_tag ‘byebye_js’ %>`



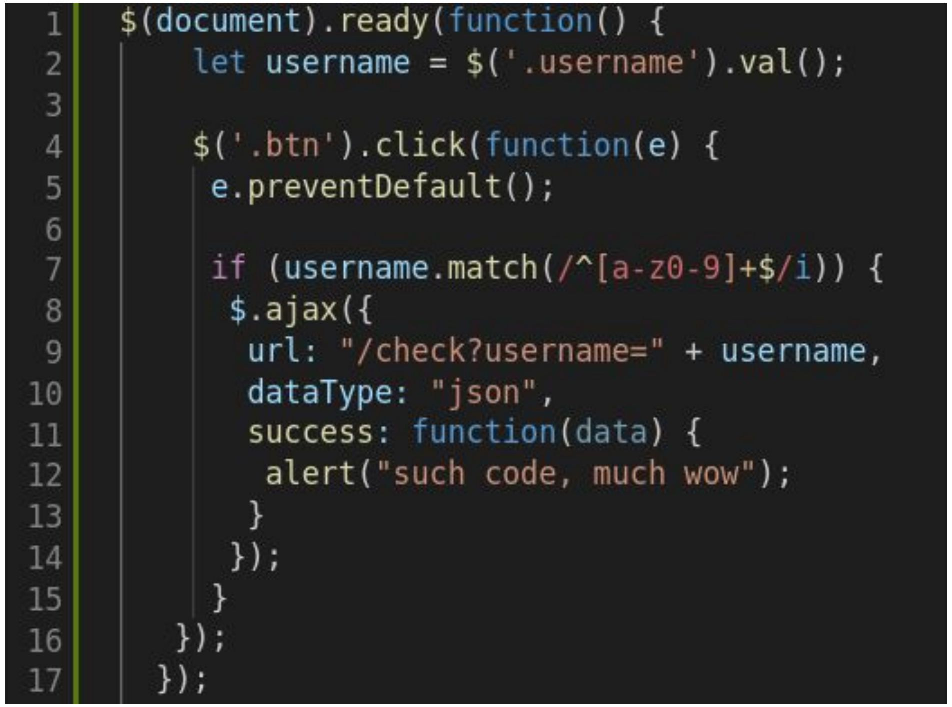
```
app
  assets
  channels
  controllers
  helpers
  javascript
    channels
    packs
      application.js
      sample1.js
      sample2.js
  jobs
  mailers
  models
  views
bin
config
db
lib
log
node_modules
public
storage
test
tmp
vendor
.browserslistrc
.gitattributes
.gitignore
.ruby-version
babel.config.js
config.ru
Gemfile
Gemfile.lock
```



```
app
  assets
    config
    images
    javascript
      sample1.js          U
      sample2.js          U
    stylesheets
  channels
  controllers
  helpers
  javascript
  jobs
  mailers
  models
  views
bin
config
db
lib
log
node_modules
public
storage
test
tmp
vendor
.browserslistrc
.gitattributes
.gitignore
.ruby-version
babel.config.js
config.ru
Gemfile
Gemfile.lock
```

Practice Exam Question

You are trying to create an instagram username checker where users can input a preferred username and see if it's available. Which of these choices correctly describes what this code does? **Note** that a username is considered valid if it only contains alphanumeric characters and is unique.

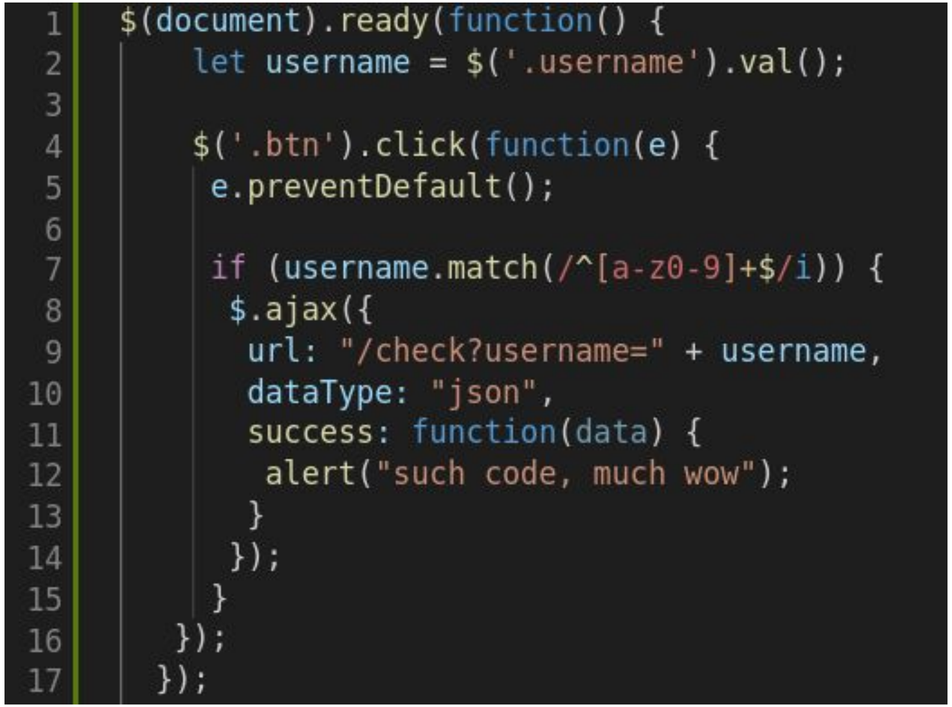


```
$(document).ready(function() {
    let username = $('.username').val();

    $('.btn').click(function(e) {
     e.preventDefault();

     if (username.match(/^[a-z0-9]+$/i)) {
      $.ajax({
       url: "/check?username=" + username,
       dataType: "json",
       success: function(data) {
        alert("such code, much wow");
       }
      });
     }
   });
 });
```

- ❏ Validates the username and makes sure it's longer than 3 letters.
- ❏ Only makes an AJAX call with a valid username
- ❏ Makes an AJAX call and if successful displays a pop-up with "such code, much wow"
- ❏ None of the given options

Practice Exam Question

You are trying to create an instagram username checker where users can input a preferred username and see if it’s available. Which of these choices correctly describes what this code does? **Note** that a username is considered valid if it only contains alphanumeric characters and is unique.



```
$(document).ready(function() {
    let username = $('.username').val();

    $('.btn').click(function(e) {
     e.preventDefault();

     if (username.match(/^[a-z0-9]+$/i)) {
      $.ajax({
       url: "/check?username=" + username,
       dataType: "json",
       success: function(data) {
        alert("such code, much wow");
       }
      });
     }
   });
  });
```

- ❑ Validates the username and makes sure it’s longer than 3 letters.
- ❑ Only makes an AJAX call with a valid username
- ❑ Makes an AJAX call and if successful displays a pop-up with “such code, much wow”
- ❑ None of the given options

Attendance Form

https://tinyurl.com/cs169-disc-6

Passcode: webpacker